Naval Research Laboratory

Washington, DC 20375-5000

NRL/MR/5524—92-6944

Development of a Fault Isolation System Database Manager

C. B. BARCLAY AND J. A. MOLNAR

Communications Systems Branch
Information Technology Division

February 10, 1992

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 10, 1992	Final (1990 to present)

4. TITLE AND SUBTITLE

Development of a Fault Isolation System Database Manager

5. FUNDING NUMBERS

NIF
WU - DN580-058

6. AUTHOR(S)

C. B. Barclay and J. A. Molnar

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5524—92-6944

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Air Engineering Center
Lakehurst, NJ 08733

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Fault Isolation System Database Manager (FISDM) is an interactive database interface developed to enhance data and configuration management of causal models. A causal model of a Unit Under Test (UUT) describes the interrelationship of the electronic components composing the UUT. The interface, written in C, was integrated with the Empress data management software package to access data manipulation routines. Standard data manipulation features are incorporated to enhance the creation and management of the knowledge database. Output features allow delivery of the data in multiple formats. The format required for the Fault Isolation System (FIS) knowledge database is produced directly as an output option. Additionally, a comma delimimted format compatible with other database and spreadsheet software provides portability and flexibility for configuration management. Data management is achieved through maintenance of a standard format which can be translated to working version in other formats. FISDM permits central management of expert systems that require occasional modifications from multiple sources. This is a situation which often occurs over the fielded lifetime of military electronic systems.

14. SUBJECT TERMS		15. NUMBER OF PAGES
Expert system	Fault isolation	71
Maintenance aid	Knowledge engineering	16. PRICE CODE
Database management	Human interface	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

CONTENTS

1.0 INTRODUCTION 1

1.1 Format Description 1

2.0 OPERATIONAL OVERVIEW 1

3.0 THE EMPRESS RDBS 2

3.1 Error Checking in FISDM 2

4.0 FISDM SUPPORTED DATA FORMATS 2

4.1 Ascii Text Format 3

4.2 Comma Delimited Format 4

4.3 FIS Format 4

4.4 FISDM Format 5

4.5 Data Element Format 5

4.5.1 Rule Data Elements 6

4.5.2 Test Data Elements 6

5.0 FISDM OPERATIONAL STRUCTURE 6

5.1 Main Menu 6

5.2 Conversion Menu 6

5.2.1 Conversion from ASCII text to FISDM 6

5.2.2 Conversion from Comma Delimited to FISDM 7

5.2.3 Conversion from FISDM to Comma Delimited 7

5.2.4 Conversion from LISP to FISDM 7

5.2.5 Conversion from FISDM to LISP 7

5.3 View Menu 7

5.3.1 Indexing a Table 7

5.3.2 Sorting a Table 8

5.3.3 Search for a Data Element 8

5.3.4 Display Table 8

5.3.5 Add Record 8

6.0 DEMONSTRATION OF DATA INTERCHANGE - MACINTOSH EXAMPLE 8

6.1 Transferring Data to a Personal Computer 9

6.1.1 Macintosh Example 9

6.1.2 Macintosh Panorama Environment 9
6.1.3 Saving Modifications Made In Panorama 9
7.0 SUMMARY 10
REFERENCES 10
APPENDIX A - INSTALLATION PROGRAM 13
APPENDIX B - TUTORIAL 17
B.1 INTRODUCTION 17
B.2 MAIN MENU 17
B.2.1 Printing 17
B.2.2 Database Directory 18
B.2.3 New Database 18
B.2.4 Quit FISDM 19
B.3 CONVERSION 19
B.4 VIEWING 20
B.4.1 Indexing 20
B.4.2 Sorting 21
B.4.3 Database View 21
B.4.4 Search 23
B.4.5 Addition 23
B.4.6 Return to Main Menu 24
APPENDIX C - FISDM.C PROGRAM LISTING 25

FIGURES

Fig.		Page No.
1	Excerpt from ASCII rule format	3
2	Excerpt from ASCII text format	3
3	Rule information as stored in comma delimited format	4
4	Test information as stored in comma delimited format	4
5	FIS format for rules and test information	5
6A	Rule data structure	5
6B	Test data structure	5
7	Panorama dialog box	10
B1	Main Menu	17
B2	Conversion Menu	19
B3	View Menu	20
B4	Sample sort of rule database	21

Development of a Fault Isolation System Database Manager

1.0 INTRODUCTION

The Fault Isolation System (FIS) is an expert system shell developed by the Navy Center for Applied Research in Artificial Intelligence (NCARAI). It is a diagnostic reasoning system used to diagnose faults in electronic systems employing analog circuitry. The FIS provides a generic interface engine that operates on individual knowledge databases. These knowledge databases uniquely describe the electronic system as a model of causal relationships. The FIS effectively addresses the knowledge representation aspect of knowledge acquisition, however, neither the translation of knowledge to the appropriate form nor the management of the translation process is adequately considered. Since a knowledge base is a dynamic entity that must remain a current representation of knowledge, a knowledge acquisition preprocessor, FIS Database Manager (FISDM), was developed to address these issues. FISDM facilitates creation, modification, and translation of the knowledge database over the lifetime of the data. FISDM is software written in C that interfaces with the Empress Relational Database Management System software package to provide data manipulation capabilities for FIS knowledge representation requirements. FISDM is currently implemented on a SUN 4/330 workstation, but it permits knowledge acquisition through multiple computer platforms and general purpose software.

1.1 Format Description

Several unique formats are used in the text to help improve readability of the manual. These formats remain standard throughout the manual.

Italics are used when a filename, variable name, or important word or phrase appears in the paper.

Bold print is used to display keyboard entries made by the user. ***Italicized bold*** identifies variable names that the user must enter. An example would be ***filename***, where the user enters the filename particular with their system. In many cases, interaction with the computer is described in the text. The font `Monaco` indicates these interactions. As above, **`bold`** `Monaco font` refers to user responses and ***`italicized bold`*** `Monaco font` identifies variable names the user must enter.

2.0 OPERATIONAL OVERVIEW

The purpose of FISDM is to simplify the creation and modification of the knowledge database. The program includes features not included in the FIS editor, such as data tables for viewing and the ability to perform complex sorting. Three types of user menu features are accessible through the interface. The first type is a main menu that performs the rudimentary Empress functions of printing, creating new databases, and viewing the names of existing databases. The second is a conversion routine menu for converting between different formats. The third is a view menu that uses Empress functions to facilitate sorting and inspection.

Manuscript approved December 10, 1991.

The program is a menu driven database manager, capable of maintaining complex expert system knowledge databases. Its usefulness is derived from its powerful yet simplified data manipulation functions. Interactions with the knowledge database are processed through user responses to program queries. The program is responsible for data formatting and error checking. Once formulated, knowledge database file maintenance is performed by software commands. The translation to the knowledge base format can be accomplished from any of three user data formats - FIS format, comma delimited format and ASCII text format. The user may use any of the three formats to create the knowledge database while taking advantage of the data manipulation and presentation functions available in FISDM. The intrinsic elements of the program provide the user with a capability to look at the data in a style that is comprehensible, e.g., data elements are easily associated with the respective data. FISDM presents data records in a chart form that identifies the data in the record with the appropriate element. The program acts as an interface between the user and the knowledge base. FISDM acts as a tool to perform the essential tasks involved in upkeeping knowledge bases, while facilitating the user's access to the data. FISDM creates an efficient central management system.

3.0 THE EMPRESS RDBS

Empress is a relational database management and application development system. Empress is able to perform traditional database functions while incorporating custom features. There are two ways to invoke commands in Empress: Query Language and MX commands.

The Query language is an interactive language with an English-like syntax for database access. Query language commands can be accessed outside FISDM either by entering the Empress command line interpreter or in a C program through a gateway to the Empress program. Some commands, such as deleting databases, are available through this option but are not provided in the FISDM program.

MX commands are accessed as C language functions by the C compiler when the header file *mscc.h* is included. MX commands are functions in a C program that correspond to commands in Empress. MX commands are faster and more efficient than the Query language commands. For this reason the FISDM program primarily uses MX commands to access Empress.

FISDM uses only a small subset of the commands available through Empress. The functions selected for inclusion in FISDM represent only the basic functions determined to be essential to knowledge acquisition and management. The determination was made based upon extensive experience in creation and management of FIS knowledge bases associated with the AN/SQS-53B Sonar System, and the Technician's Assister System[1,2]. Further FISDM functionality can be accessed through the Empress Query language. For more information on additional Empress Query language commands, the reader is referred to the Empress User's Manual[3].

3.1 Error Checking in FISDM

The program's interface with Empress provides it with built-in functions to alert the user of problems, e.g., an incorrect file name. The variable *mxerrt* is responsible for toggling this feature. When the variable is set to 1, the program is instructed not to terminate when an error arises but to follow instructions coded into the program. When there are no instructions for the program, execution is halted and a message will come to the screen saying: ***** ERROR TYPE ***** followed by an explanation of the error. One type of error is a user error, usually caused by passing incorrect table or file names to the database. For instance, one could expect this error if instead of typing in *tests* for a test database, *rules* was typed in. Another type of error is a database error. This occurs when the database is found to be corrupted because of hardware or software problems. If this happens, consult the Empress user manual for further instructions.

4.0 FISDM SUPPORTED DATA FORMATS

In the early stages of expert system development it was found that constructing the causal model with the FIS editor was tedious. One programming goal was to make the FIS data structure

accessible to individuals with little or no knowledge of FIS or LISP. The data format used by FIS is in a LISP compatible syntax. During the implementation of FIS[1], knowledge database construction was performed with text editors and database programs; the use of the FIS editor was used only as a secondary source for knowledge construction. As a result, the FIS data format was not used in the knowledge acquisition phase of the expert system development. Two other formats evolved for knowledge acquisition: ASCII text format and comma delimited format. FISDM supports the implementation of these two formats, the FIS Lisp format, and a format distinct to FISDM.

4.1 ASCII Text Format

The ASCII format arose because FIS and the FIS editor were not available to the knowledge engineer when the knowledge base was first under construction. The ASCII format was suitable for construction with any text editor. In addition, the ASCII format was more accessible for formulation and debugging since it presented the data format in clearly defined columns that required no special knowledge of FIS or LISP. The ASCII format implemented strict rules for ordering and spacing. The format was not compatible with the LISP format required by FIS, requiring software to extract the data and place it in a proper FIS format. Occasionally data is entered improperly resulting in conversion errors. The conversion routines initially developed are implemented in FISDM. FISDM converts all the text data that fulfill the format requirements and identifies the line numbers of nonconforming data. Depending upon the magnitude of conformity and the size of the data file, the user has the option of altering the text file and converting again or simply entering the nonconforming data directly into the FISDM database.

FISDM provides a one-way conversion capability for the text format. Once data is converted from this format it may not be converted back. FISDM maintains the display style of the ASCII format by allowing a function that emulates this style for printing. The formats are not compatible and attempts to read the FISDM print format during an ASCII to FISDM conversion would cause data to be improperly added to the FISDM database.

The ASCII format required distinct files for rules and tests. Each file had distinct rules regarding its structure. The ASCII format structures for the rule and test files are described in Reference [1]. A sample of the ASCII text format for the rule file is displayed in Fig. 1. A sample of the test data in ASCII format is shown in Fig. 2.

```
Module:  a26a1a1_delay_line

No  Cause                               Effect                         Type      Precondition
1   a26a1a22_a17-out beam_1_wave bad    a26a1a1J4 bit_0_left_input bad  s         t
2   a26a1a22_a17-out beam_2_wave bad    a26a1a1J4 bit_0_left_input bad  s         t
3   a26a1a22_a17-out beam_3_wave bad    a26a1a1J4 bit_0_left_input bad  s         t
```

Fig. 1 - Excerpt from ASCII rule format

```
                                                  $3
           $1                                     POSS
           TEST       $2                          QUAL  $4   $5    COST,
  NAME     POINT      PARAMETER        UNITS      RES   MIN  MAX   SECONDS PREREQUISITES       NAME

A1A1J4   a26a1A1J4   logic_levels     logic      bad               20      scope ready         logic_1
                                                 ok                        a26a1_drawer open
                                                                           unit-26_door open [1
                                                                           probe_on a26a1A1J4
```

Fig. 2 - Excerpt from ASCII test format

4.2 Comma Delimited Format

The development of FISDM anticipated that knowledge engineers would use databases and spreadsheets on other computers to perform FIS knowledge acquisition. These programs store data in their own unique format; most also have an option to export data in a standard format of tab delimited or comma delimited. The comma delimited format was incorporated into the FISDM program to increase portability between machines. The tab delimited format was not selected because tabs are non-printing characters for most text editors, causing confusion when viewing and editing files created in such a format. The comma delimited format consisted of viewable data separators allowing tables both easy to read and modify.

When importing from the comma delimited format, some empty data fields may be filled by default values. These data fields and values are, respectively: frate, 0.1, precondition, t, cost, 10, type, D.

As was the case with ASCII text format, the comma delimited format requires separate files for rule and test information. An example of the comma delimited format for the rules information is shown in Fig. 3. The format for the file containing the test information in comma delimited format is shown in Fig. 4.

```
module name, frate, cause, effect, type, precondition
```

Fig. 3 - Rule information as stored in comma delimited format

```
name, test point, parameter, units, qualifying values, min, max
cost, prerequisit, instruction, type, focal module
```

Fig. 4 - Test information as stored in comma delimited format

4.3 FIS Format

The FIS data format is a LISP compatible format that is used by FIS functions involving editing and compiling. As shown in a sample of this format in Fig. 5, the data file is composed of three lists that can contain graphical information followed by a rule list, a test list, and lists for preconditions, orders and instructions. In Fig. 5, there is no graphical information, preconditions, orders or instructions, thus the NIL values. A NIL represents a list that contains no information. Only the rule and test lists are absolutely essential to the functionality of FIS as an expert system for fault isolation; they are the only lists input by FISDM. The other LISP lists may either be defined as a set or an empty set (NIL) depending upon the FIS implementation. Consequently, FISDM only considers management of rule and test information; other information used by FIS cannot be managed with FISDM. If the user converts from the FIS format, all the graphical, precondition, order and instruction information are lost. Additional details on this format can be found in Reference [1] and Reference [4].

```
NIL
NIL
NIL
(((NAME a26a1a1_delay_line) (FRATE .1)
   (CAUSAL-RULES NIL)))
((A1A1J4 (a26a1A1J4 logic_levels
      S1
         ((ok (11.88 12.12)) (bad (-inf 11.88) (12.12
inf)))
         volts
      D
      10
      NIL))
NIL
NIL
NIL
```

Fig. 5 - FIS format for rules and test information

4.4 FISDM Format

FISDM uses data standards to express the information contained in data records. Two files, *rules_stand* and *test_stand*, must be located in the directory when FISDM is invoked. The FISDM installer, as described in Appendix A, builds these files automatically. The standard files provide Empress with the specifications on how to create new database tables. Figures 6A and 6B present the data structures established for the respective rule and test data. Changing the content of any field in the database requires the files be modified to reflect that change.

module	char	40
frate	char	06
cause	char	50
effect	char	50
type	char	10
precond	char	10

Fig. 6A - Rule data structure

name	char	20
test_point	char	20
parameter	char	30
units	char	15
qual1	char	20
min	char	08
max	char	08
cost	char	05
prereq	char	100
instruction	char	30
type	char	30
focal module	char	05

Fig. 6B - Test data structure

4.5 Data Element Content

One feature that is consistent across the four formats is the data. In each format there exists at minimum two types of data, called the rules and the tests. Several more data types exist in the FIS format. The purpose of FISDM is to transpose the data elements across the various formats and

maintain the integrity of the data. A brief description of the data sets is presented in sections 4.5.1 and 4.5.2, while an expanded discussion of FIS data elements is found in Reference[1].

4.5.1 *Rule Data Elements*

Rules are composed of module names, causes, effects, preconditions, type and failure rate (FRATE). Type is not currently used by FIS. FRATE is established from the normalized mean time to failure information for hardware components. The FRATE defaults to the value of 0.1 if the actual value is not immediately known.

Both causes and effects are composed of subelements. Causes may be a single subelement or a group of three subelements. If it is a single subelement, the subelement should be consistent with an existing module name in the rule data set. If it is a three subelement group it is composed of a terminal, a parameter, and a definition of the malfunction state, e.g., *hi*.

Effects are composed only of a three element group. The content of the subelement group is the same as the content of the three subelement groups of the cause data element.

4.5.2 *Test Data Elements*

FIS tests are composed of nonoptional and optional data. The required data are test name, test point, parameter, test. Several items are optional: units, minimum quantitative value, maximum quantitative value, prerequisites, instruction name, and type. Test is a required item for FIS, but if this item is not known a default value of *diagnostic* is supplied.

Qualitative values are often composed of several subelements. Each subelement expresses a possible value for that test.

5.0 FISDM OPERATIONAL STRUCTURE

FISDM is composed of a hierarchy of menu functions, a main menu and two specialized submenus that are accessed from the main menu.

5.1 Main Menu

When FISDM software is executed, a main menu is presented to the user. The main menu presents access to the functions of printing, database directory listing and creating a new database. In addition, the main menu provides access to data format conversion and database modification menus. FISDM creates files through the printing command in this menu that are similar to the ASCII text format. These files may be printed through printing commands provided in Unix.[5] File contents are displayed in a 132-column format. A database directory listing is provided to allow the user to identify currently available databases. Creating a database forms new databases in the format specified by the user. The user adds the first record by responding to questions given by FISDM.

5.2 Conversion Menu

The conversion menu allows the user to convert among the three file formats used with FIS: ASCII text, comma delimited, and LISP compatible. The only conversion not supported is from FISDM to ASCII text since this format is not helpful for data retention or management but exists from the early development stage of the AN/SQS-53B, TAS[1, 2].

5.2.1 *Conversion from ASCII text to FISDM*

This function will convert either a rule or test file into the database. Data is accepted from a file that is divided into columns according to data records. Any data that is not in the proper columns will be identified by FISDM as an error. The user may either choose to modify the offending line and reconvert or add the record manually. As discussed previously, the program does not provide a capability to convert back to this format.

5.2.2 *Conversion from Comma Delimited to FISDM*

ASCII comma delimited text is a common option for many microcomputer database software programs. Data prepared with such a program in the formats described for comma delimited rule and test data in Section 4.2 are accessible to FISDM. Invoking the option to convert from comma delimited to FISDM provides this access. Each data line is read into either rule or test format as defined by the user. Data delimited by commas is matched with the appropriate field by the database manager. Since the program uses commas to separate the fields, using a comma as a character in the data would cause information to be misplaced. Files should always contain the proper amount of commas, even if a field is empty. Warnings are given if a line has too few or too many commas.

5.2.3 *Conversion from FISDM to Comma Delimited*

To support the remote development of knowledge databases, FISDM provides a function to convert to comma delimited format. This allows database software on other computers to use data that is stored in FISDM. This capability acknowledges that users may rather work on software with which they are familiar. This provides an easy way to convert back and forth from FISDM to take advantage of the user's knowledge of another spreadsheet, database or software package and the power of the FISDM centralized database manager. An example of data manipulation with software on a personal computer is provided in Section 6. This example displays interactions with the Panorama program on a Macintosh computer.

5.2.4 *Conversion from LISP to FISDM*

The FIS editor produces output in a LISP readable format, however this format is difficult to load into a text editor for viewing and modification. FISDM was developed to provide a capability for viewing and manipulating the FIS knowledge base.

When the FISDM function to convert from LISP format is invoked, the file is first preprocessed. The preprocessor removes all carriage returns in the file to create one long variable. It also modifies the LISP file to replace NILs with the empty set (). In performing the conversion FISDM uses C functions to emulate the action of the LISP functions *car* and *cdr*. Users are alerted that the conversion may take several minutes.

5.2.5 *Conversion from FISDM to LISP*

The most important function provided by FISDM is the ability to convert files to the FIS LISP format. The actual file produced is the FIS *.v* file. In this format the file can be viewed with the FIS editor. The function takes the rule and test files that were loaded together and exports them into the single FIS *.v* file. All rules and tests are grouped into their respective LISP lists.

5.3 View Menu

The view menu is the second submenu accessible from the main menu. The user must know the table name and whether it is in rules or tests format before entering this menu. Modification, viewing and general file upkeep functions are provided through the view menu.

5.3.1 *Indexing a Table*

Indexing a table allows searches and sorts to proceed faster through creation of a lexically ordered listing in memory. After a table is indexed, further calls to index result in a prompt asking the user to delete previously declared indexes or proceed with additional indexing. The option to delete previous indexes is provided because indexes occupy as much memory as the original listing. Continued indexing reduces the amount of RAM available for other operations and slows processor executions as virtual memory must be more frequently accessed. Deleting an index can save room for other operations. All indexes are deleted upon termination of FISDM.

5.3.2 *Sorting a Table*

The user may sort any table by an appropriate field name and output the result to a file or the screen. The sort may be either ascending or descending. Output to the screen is not recommended for long files, as screen paging is not provided. Tables may be sorted by multiple fields with each field name separated by only commas, no spaces. The primary field to be sorted should be placed first, with all others following in the order of preference.

5.3.3 *Search for a Data Element*

This function allows the user to locate a specific data element within a field name for a rule or test table. The user is asked to supply the field name for the data and the data element. The function searches the table and displays all matches. The display consists of the table headings followed by the complete records of all matching data elements. Only the table headings are displayed if no match is located. The search is case sensitive and the data element match must be exact. Matching a truncated or partial data element is not supported.

5.3.4 *Display Table*

This function provides the capability to sequentially display the individual records of a table and to perform minor maintenance. The records are displayed in a standard database form, with each data field preceded by the respective field name. A command line follows the record presentation. The following functions are available at the command line: move forward or move backward one record in the table, skip forward in the table, modify the current record, delete the current record, and return to the View Menu.

The forward and reverse movement capability is self-explanatory.

The skip command allows the user to either skip a certain number of records ahead, or move to a specific record. A user may request a specific record by giving the name of that record. At the first occurrence of the given name, FISDM will stop. If the database does not contain a record with the specified name, or the user requests to move forward a number higher than the amount of records left, FISDM returns to the View menu.

The modify command allows the user to modify a single field in the data record being viewed. More than one field in the data record may be modified by selecting the modify command until the record changes are completed.

The delete command deletes the current record being viewed. There is no capability to undelete, so this command should be used cautiously.

5.3.5 *Add Record*

This function is called by both the Main Menu and the View Menu. In the Main Menu after creating a new table, this is used to add the first record. In the View Menu it is used to append a record to an existing database. FISDM prompts the user for the data elements of the record, and positions the data properly into the table. After the user confirms that the information is correct, FISDM adds the record.

6.0 DEMONSTRATION OF DATA INTERCHANGE - MACINTOSH EXAMPLE

One goal in the development of FISDM was to permit FIS knowledge base construction in an environment chosen by the knowledge engineer. FISDM meets that goal by providing an interface for knowledge database construction on personal computers. This was demonstrated by using FISDM to exchange files with files prepared with the Panorama database software package for the Macintosh.

6.1 Transferring Data to a Personal Computer

Many commercial and shareware software programs are available to transfer data between computers. Depending on the hardware and software configuration, these work over telephone lines in conjunction with a modem, over networks, or with direct connections between computers. Remote connections through a modem require that each computer has an accessible modem and available phone line. Baud rates and software protocol at each computer must match. Access through a network requires each computer to exist at a network node that is accessible from the other computer. If this condition is established then the specific compatible software protocol should allow execution of the data transfer. File transfer through a direct connection between computers is similar to the network case. Proper hardware connections must exist between compatible I/O ports of each computer. Compatible software protocols then execute the file transfer.

6.1.1 *Macintosh Example*

In this example the Macintosh and Sun computers reside as nodes on an Ethernet LAN. The Sun operating system allows network access through the use of TCP/IP protocol. NCSA Telnet is used on the Macintosh to implement the use of TCP/IP. The user must possess a username and password for both the Sun and the Macintosh.

The user who accesses the Sun from the Macintosh must first establish a Telnet session on the Sun and login. The user then pulls down the *Network* menu and selects the *Send FTP Command*; this results in the Sun sending the FTP command to the Macintosh. When the connection is established, the username and password on the Macintosh must be supplied. The connection is then available for file transfer. The user who accesses the Macintosh from the Sun must be certain that NCSA Telnet is running on the Macintosh, otherwise the Macintosh will not respond to connection requests.

After the connection is established and the user logs in, file transfer is accomplished with the *get* and *put* commands in FTP. *Get* transfers the specified file from the remote host to the local host and places the contents in a file. If a new filename is not specified by the user on the command line, the contents will be placed in a file of the same name. *Put* transfers the specified file from the local host to the remote host and places the contents in a file. Here again, if a new filename is not specified by the user on the command line, the contents will be placed in a file of the same name. Refer to Unix documentation for further Telnet or FTP command information [5].

6.1.2 *Macintosh Panorama Environment*

When starting the Panorama application, the user is asked to identify and open a file. Choices in the window ask whether the file will be in Panorama, OverVUE, or Import format. Click the mouse on the Import button and find the file to be transformed. This will load the database into Panorama. Once the data is imported, the user may perform any valid Panorama functions such as change column names, widths, or insert columns [6]. However, the user must remember that any changes in size of the data fields or adding new data fields are not possible without modifying FISDM and FIS. Changes to the data content can be made without effect on FISDM, although comma use in data is verboten because of formatting procedures.

6.1.3 *Saving Modifications Made In Panorama*

After making changes to the data, select *Save As...* from the File Menu. A dialog box is presented and the user supplies a name for the file. The data may be stored in Panorama or Text Only formats. If the data is to be exported to FISDM then the Text Only format is selected. Click the *Save* button and a second dialog box is presented, as shown in Fig. 7. Click on the *Select All* button. This ensures that information in all columns will be saved. Commas should be selected to ensure FISDM compatibility. Press the *OK* button to save the data.

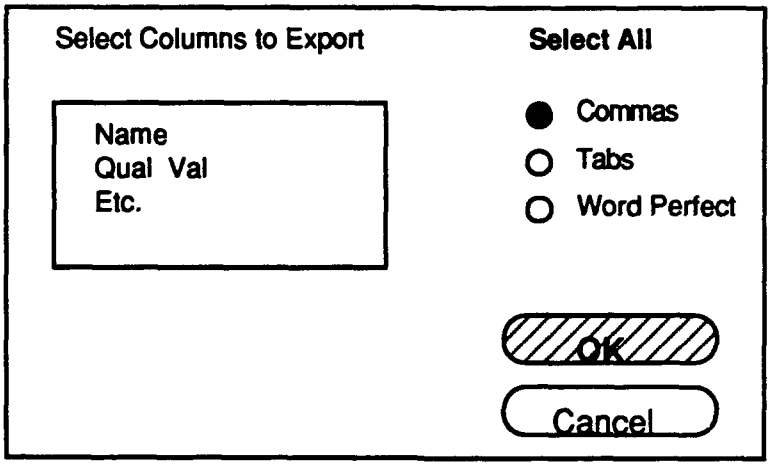

Fig. 7 - Panorama Dialog Box

The result of these steps produces a FISDM compatible file suitable for transfer. Details of Panorama usage are referred to in Reference [6]. The selection procedure may be different for other personal computer based software.

7.0 SUMMARY

FISDM software was developed to enhance the acquisition of causal and test information for the FIS expert system shell. The software interfaces with the Empress database software package to provide data manipulation functions. FISDM software provides the user interface for creating, modifying and using knowledge base information. FISDM produces the knowledge base in the format appropriate for the FIS shell. Additionally, FISDM provides an interface to direct and redirect knowledge base data to a generalized format common to many database software programs.

FISDM was demonstrated to accept accurate data directly from a user, from ASCII text, from other database programs that produce output in comma delimited format, and from an existing FIS knowledge base. FISDM also demonstrated the ability to produce correctly a comma delimited text file suitable for use with database software.

FISDM expands the options available for knowledge base construction by accepting multiple formats and allowing the construction of knowledge bases to be a distributed function that can be centrally managed. FISDM is expected to be most useful for groups using FIS as a fault isolation expert system on equipment requiring a knowledge base for each. First applications of FISDM will be for generation of fault trees for automatic test equipment test program sets, e.g., the Navy CASS program.

REFERENCES

1. J. A. Molnar, "Creation, Validation, Testing and Data Management of a Knowledge Base Designed for a Technician's Assister System for the AN/SQS-53B, Unit 26, Using a Fault Isolation System," NRL Report 9296, December 17, 1990.

2. J. A. Molnar, C. F. Sarteschi, "Validation of the Technician's Assister System," Proceedings of the IEEE Conference on Managing Expert System Programs and Projects, September 10-12, 1990, p.p. 201-204.

3. Empress Software, Inc., *Empress User's Manual, Version 2.4*, (Empress Software, Inc., Greenbelt, MD, 1987).

4. W. M. Spears, "FIS Users Manual," Memorandum, NRL Center for Applied Research in Artificial Intelligence, Washington, DC, March 1989.

5. Sun Microsystems, Inc., *Sun Operating System Reference Manual*, Release 4.0, (Sun Microsystems, Inc., May 9, 1988).

6. ProVUE Development Corp., *Panorama User's Guide, Version 1.1*, (ProVUE Development Corp., Huntington Beach, CA, 1988).

APPENDIX A: INSTALLATION PROGRAM

The FISDM program installer was created to facilitate the installation of FISDM. The installation program will only work on SUN workstations, as some of the C commands are intrinsically part of the SUN subset. The installation of FISDM requires two files: *fisdm.c* and *fisdm_install.c*. In addition, the directory from which *fisdm_install.c* is executed from must not contain a subdirectory named *database*. The installation program will create Empress files in this directory. Compile the installation program, *fisdm_install.c*, by typing *cc fisdm_install.c*. The executable file that results may be run by typing *a.out*. The installation program will check that all necessary files are present, and will build any that are not found. These files include *fisdm.h*, *rules_stand*, and *test_stand*. The first time the FISDM program is installed on a system, it is best to let the installation program build the files as it adds commands specific for the individual workstation. The executable file FISDM may be moved to other directories after the installation program compiles the file *fisdm.c*. The executable file name is *fisdm*. The following code is a listing of the program *fisdm_install.c*.

```
/*

              The Fault Isolation System Database Manager (FISDM)
                                Installation Program
       Developed by Code 5524, Naval Research Laboratory, Washington DC 20375

   Author:  Chris Barclay
   Date:    July 8, 1991
   Object:  This program installs fisdm and all necessary files.
   Name:    fisdm_install.c (FIS Database Manager Installer)
   Version: works on Sun OS 4.1 and 4.1.1, Empress 4.5

   barklay@itd.nrl.navy.mil
*/

#include <stdio.h>
#include <sys/param.h>

#define  LINE_LENGTH    300

FILE *fopen(), *fp1;

char a, cwd[900], new[1000];

main()

{
  system("clear");
  getwd(cwd);   /*this gives the pwd*/
  if ((fp1=fopen("fisdm.c", "r+"))==NULL)
  {
    printf("**ERROR** fisdm.c must be located in the same directory.\n");
    exit(0);
  }
  fclose(fp1);
  printf("\n\n\nWelcome to the FISDM installer.\n\n");
  if ((fp1=fopen("rules_stand", "r+"))==NULL)
  {
    fp1=fopen("rules_stand", "w");
    fprintf(fp1, "/zeno/barclay/database|rule|module|char|char|40|1|0|0|n\n");
    fprintf(fp1, "/zeno/barclay/database|rule|frate|char|char|6|1|0|0|n\n");
    fprintf(fp1, "/zeno/barclay/database|rule|cause|char|char|50|1|0|0|n\n");
    fprintf(fp1, "/zeno/barclay/database|rule|effect|char|char|50|10|0|n\n");
    fprintf(fp1, "/zeno/barclay/database|rule|type|char|char|30|1|0|0|n\n");
    fprintf(fp1, "/zeno/barclay/database|rule|precond|char|char|30|1|0|0|n\n");
    printf("Installing file rules_stand.\n");
    fclose(fp1);
  }
   else
  {
    printf("Rules data format already installed, going to next step.  Check to
make sure\n");
    printf("the file rules_stand is the correct version.  Otherwise, delete the
file\n");
    printf("and re-run program.\n");
  }
  if ((fp1=fopen("test_stand", "r+"))==NULL)
```

```
  {
    fp1=fopen("test_stand", "w");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|name|char|char|20|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|test_point|char|char|20|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|parameter|char|char|30|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|units|char|char|25|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|quall|char|char|20|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|min|char|char|8|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|max|char|char|8|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|cost|char|char|50|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|prereq|char|char|100|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|instruction|char|char|30|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|test|char|char|30|1|0|0|n\n");
    fprintf(fp1,
"/zeno/barclay/database|test_standard|type|char|char|30|1|0|0|n\n");
    printf("Installing file test_stand.\n");
    fclose(fp1);
  }
   else
  {
    printf("Test data format already installed, going to next step.  Check to make
sure\n");
    printf("the file test_stand is the correct version.  Otherwise, delete the
file\n");
    printf("and re-run program.\n");
  }
  printf("Making the Empress directory %s/database.\n", cwd);
  system("msmkdb database");
  a= 0x22;
  strcat(cwd, "/database");
  strcpy(new, "#define  DATABASE "); /*this is the changed line in fisdm.c*/
  new[strlen(new)]=a;
  strcat(new, cwd);
  new[strlen(new)]=a;
  if ((fp1=fopen("fisdm.h", "r+"))==NULL)
  {
    fp1=fopen("fisdm.h", "w");
    printf("Making the fisdm.c header file fisdm.h.\n");
    fprintf(fp1,
"/*****************************************************************\n");
    fprintf(fp1, "*         This header file goes with the software fisdm.c
*\n");
    fprintf(fp1, "*         Developed by Christopher Barclay, Code 5524 NRL
*\n");
    fprintf(fp1, "*         June 18, 1991                              fisdm.h
*\n");
```

```
    fprintf(fp1,
"*************************************************************************/\n\n\n\n\n\n")
;
    fprintf(fp1, new);
    fprintf(fp1, "\n#include <mscc.h> /* this automatically calls <stdio.h>
*/\n");
    fprintf(fp1, "#include <malloc.h>\n");
    fprintf(fp1, "#include <sys/file.h>\n");
    fprintf(fp1, "#include <sys/types.h>\n");
    fprintf(fp1, "#include <sys/stat.h>\n");
    fprintf(fp1, "#include <string.h>\n");
    fprintf(fp1, "#include <ctype.h>\n");
    fprintf(fp1, "#include <time.h>\n");
    fprintf(fp1, "#include <stddef.h>\n");
    fclose(fp1);
  }
  printf("Compiling file fisdm.c...\n");
  system("mscc -g -o fisdm fisdm.c");
}
```

APPENDIX B: TUTORIAL

B.1 INTRODUCTION

The following is intended to help acquaint the user with different facets of the FISDM program. This tutorial provides a demonstration of major FISDM features. The examples follow the font format used throughout the main body of the text, as described in Section 1.1. Any command that is typed into the computer must be followed by a carriage return to begin execution.

B.2 MAIN MENU

FISDM is started by typing *fisdm*, which is the executable result of the installation as described in Appendix A. This will load the program into memory and display the menu as seen in Fig. B1.

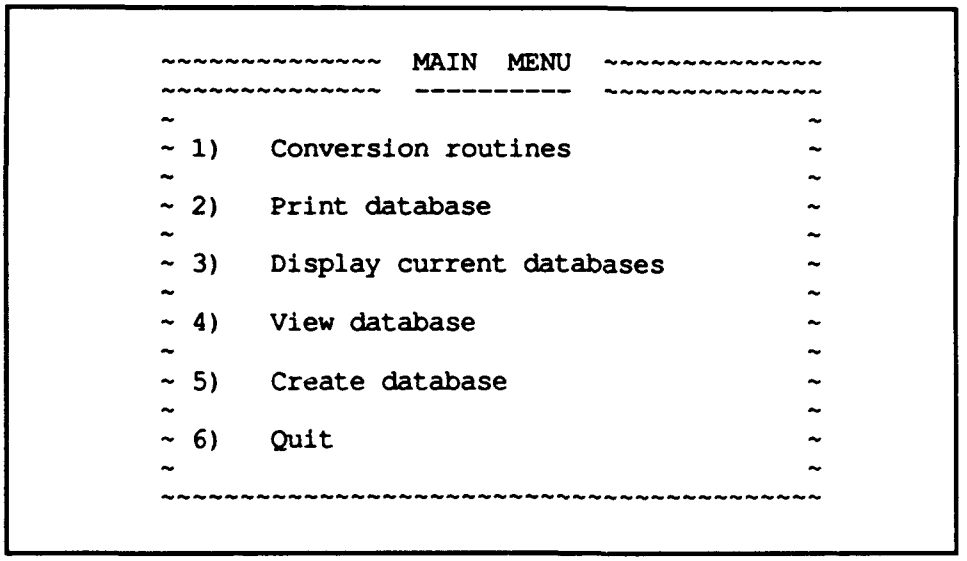

Fig. B1 - Main Menu

Selection of menu items 1 and 4 results in a new menu presentation. The menu items 2, 3, 5, and 6 perform tasks at this menu level.

B.2.1 Printing

Selection of menu item 2 takes the user to a printing function that produces an output file suitable for printing. The following questions are presented to the user:

Menu Item 2

```
What file do you want printed? right-cor86
Is it rules or tests? rules
What do you want to name the print file? rc-86
```

In this example, *right-cor86* is the name of a valid Empress database file in the user's current working directory. The type of data in the file is specified as rule data. It is on this line that the user specifies an output format that conforms to the data type. The user also identifies a new name for the output file; in the example, *rc-86* is used as the name. The result of this operation is a text file in a format designed for the type of data. Printing of the file is performed in the UNIX shell with standard UNIX commands appropriate for the type of printer used.

B.2.2 Database Directory

Selection of item 3 presents the names of all currently defined databases. Any time a database is created, it is added to this list. The user returns to the main menu by typing a carriage return.

Menu Item 3

```
**** Database: /home/barclay/database  ****
ruledata
testdata

Press enter when ready.
```

B.2.3 New Database

Selection of item 5 results in the creation of a new database; the user specifies the file name and type of data. A sample session using the rules file type is shown below.

Menu Item 5

```
Is the file format rules or tests? rules
What do you want to name the new file? left-cor78
What is the module name? a26j4
What is the cause? power
What is the effect? t1 volts hi
What is the type? d
What is the precondition?
What is the failure rate? 0.1

Module name: a26j4

FRATE: 0.1      Cause: power
Effect: t1 volts hi    Type: d     Precondition:

Is this correct (Y/N)? y
```

In this example, after selecting the create database item, the user is prompted for the type of data that will be contained in the database. Next the user is asked to supply a name for the new database file and for information on the first record in the database. The module name, cause, effect, type of test, precondition, and failure rate are the relevant data fields for the rule database. There is no error checking on the user entered data, so the user should be careful to supply the proper information, particularly for the cause and effect field. These fields should conform to the FIS data format which is an atom or triple for the cause and a triple for the effect. Details of their structure are found in Reference [1] or Section 4.4. If the user is uncertain of the data for a field, a carriage return can be

supplied to skip that entry. When a carriage return is entered for a field, FISDM leaves the field blank, with the exception of the failure rate which is supplied a default value of 0.1. After the user has supplied answers to the data field questions, the information is displayed. The user is asked whether the displayed data is correct. In the example above, **y** is entered, the record is added to the database, and the Main Menu is redisplayed. If **n** is entered, the user is prompted again to enter the correct information.

B.2.4 Quit FISDM

FISDM is terminated by selecting Quit, menu item **6**. Before termination, FISDM deletes all indexes and gives warnings about any corrupted databases.

B.3 CONVERSION

Selection of menu item **1** from the Main Menu opens the conversion menu. This menu contains the functions that transpose data from one data type to another. This menu allows data to be converted between ASCII text, comma delimited, FIS and FISDM formats. The conversion menu is shown in Fig. B2.

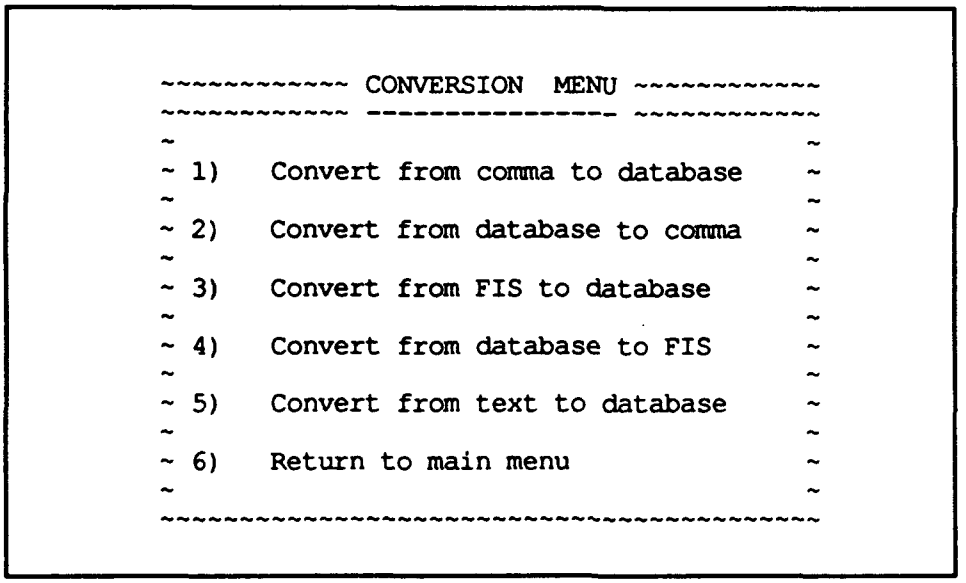

Fig. B2 - Conversion Menu

Since all menu selections result in similar interaction, a single example is presented to demonstrate the facilities of this menu.

Menu Item 1

```
Is the file format rules or tests? rules
What is the name of your rules file? rules_comma
What do you want to name the new file? rules_temp
```

In this example the selected conversion is from the comma delimited format to the FISDM database format. The user must specify whether the database is in rule or test format; in this case the rule format is used. The user must then supply the name of the file that contains the data in comma

delimited format, being certain that the format of the data file is consistent with the format previously specified. After the user enters the name of the FISDM database to be produced, the conversion is performed and the user is returned to the conversion menu.

Entering menu item 6 returns the program to the main menu.

B.4 VIEWING

Selection of menu item 4 from the main menu results in the user being prompted for the database name. After the user supplies the database name, the view menu is presented. This is used to manipulate database files and records. The view menu is shown in Fig. B3.

```
~~~~~~~~~~~~~~~~~ VIEW MENU ~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~ --------- ~~~~~~~~~~~~~~~~~~
~                                              ~
~ 1)   Create an index                         ~
~                                              ~
~ 2)   Sort a specific field                   ~
~                                              ~
~ 3)   View the database                       ~
~                                              ~
~ 4)   Search for a database entry             ~
~                                              ~
~ 5)   Add a record to the database            ~
~                                              ~
~ 6)   Return to main menu                     ~
~                                              ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fig. B3 - View Menu

B.4.1 Indexing

Selection of menu item 1 creates an index. Indexing causes the database records to be sorted respective of a specified field, e.g., module name. This provides faster retrieval in searches and reduces sort time on that field. The example below demonstrates this option.

Menu Item 1

```
What field do you want an index on:
Name, Test, Parameter, Units, Qualifying Value, Min,
Max, Cost, Instructions?
```

If the user has already created one index, the program will ask whether the old indexes should be deleted or saved. Since indexing results in a sorted database being stored in computer memory, excessive indexing will result in depletion of the available memory. As a result, unnecessary indexes should be deleted.

When at least one index exists, the sequence in the following example is followed.

```
You already have 1 index.  Do you want to D)elete and proceed or P)rocede? d
You have these indexes to delete: 1)  chris.name
```

```
Type in the number 1
What field do you want an index on:
Name, Test, Parameter, Units, Qualifying Value, Min,
Max, Cost, Instructions?
```

B.4.2 Sorting

Selection of menu item **2** sorts the database according to a specified field. The file is lexically sorted in either ascending or descending order. Output of the sorted file is directed to either the screen or a file. As indicated in Section 5.3.2, a sort on multiple fields can be performed.

Menu Item 2

```
What field are you sorting on:
Name, Test. Parameter, Units, Qualifying Value, Min,
Max, Cost, Instructions? test
Do you want to sort ascending or descending? ascending
Do you want to output to the 1) screen or a 2) file? 1
```

This example demonstrates a test database being sorted in ascending order according to the contents of the test field. The output is directed to the screen. The resulting output would be similar to that shown in Fig. B4. If the user chooses to direct the output to a file, a prompt will appear that requests the filename.

name	test_point	parameter	qual1
A1A10J1	A10J1	CIRCULATE	bad ok
A1A10J2	A10J2	RE2	bad ok
A1A10J3	A10J3	END_CLEAR	bad ok
A1A10J4	A10J4	LOAD_END_REF	bad ok
AD_CLEAR	A11J10	ATOD_CLEAR	bad ok

Fig. B4 - Sample sort of rule database

B.4.3 Database View

Selection of menu item **3** allows the user to view, modify and manipulate individual records in the database file. Initially, the first record of the database is presented on the screen along with additional options, as the following example shows.

Menu Item 3

```
Name A1A11J6
Test Point a26a1A11J6
Parameter not_ST2
Units logic
Qualifying Values bad ok Min      Max
Cost 20
Prerequisites scope ready a26a1_drawer open unit-26_door open probe_on
```

```
Instruction A11J5
Type
Focal Module

P)revious, N)ext, D)elete, M)odify, S)kip or V)iew menu?
```

Selection of the **P** option displays the record preceding the currently displayed record. If the current record is the first record of the database and this command is called, a blank record is shown.

Selection of the **N** option displays the record immediately following the currently displayed record. If the current record is the last record of the database and this command is called, the user will return to the View Menu.

The **D** option deletes the record unconditionally after the user is queried if certain that the record is to be deleted. An answer of *Y* deletes; any other answer returns the user to the record previously selected for deletion. The user should use this option with caution since there is no undeleting.

The **M** option allows a record to be modified. This modification process is similar to the creation process, as in the following example.

```
Module name: A26_CABINET_POWER

Frate: 1
Cause: A26_CABINET_POWER
Effect: A26_CABINET_PWR VOLTS BAD      Type:      Precondition: T

P)revious, N)ext, D)elete, M)odify, S)kip, or V)iew menu? m

1) Module
2) Cause
3) Effect
4) Type
5) Precondition
6) Frate
Enter the number of the field you want to modify: 6
What do you want put in its place? .1

Module name: A26_CABINET_POWER

Frate: .1
Cause: A26_CABINET_POWER
Effect: A26_CABINET_PWR VOLTS BAD      Type:      Precondition: T

P)revious, N)ext, D)elete, M)odify, S)kip, or V)iew menu?
```

S allows the user to move forward in the database either a certain number of records or to a certain module name:

```
If you want to skip a certain number of records ahead,
press 1, if you know the module name, press 2 1
How many would you like to skip? 12
```

The above example illustrates skipping 12 records. As a result, the twelfth record from the current record is displayed. Skipping by a specified number of records is common to both the rule and test databases.

If the user prefers to skip to a certain record name, the result is somewhat different for the rule and test databases. For the test database, the record name is equivalent to the test name as the

example shows. Since the test name is unique, there is no ambiguity and the record is displayed as below. Repeated calls for the same test name place the user back in the view menu.

```
If you want to skip a certain number of records ahead,
press 1, if you know the test name, press 2 2
What is the module name? AJ24B6
```

The rule database record name is equivalent to the module name. In this case there is some ambiguity since multiple records have the same module name. As a result, the first record that possesses the module name will be displayed. Repeated skips using the same module name will result with the next record having a matching module name appearing.

```
If you want to skip a certain number of records ahead,
press 1, if you know the module name, press 2 2
What is the module name? AK24_U17_FAULT

Module name: AK24_U17_FAULT

Frate: 1
Cause: A26_CABINET_POWER
Effect: A26_CABINET_PWR VOLTS BAD       Type:       Precondition: T

P)revious, N)ext, D)elete, M)odify, S)kip or V)iew menu? S

If you want to skip a certain number of records ahead,
press 1, if you know the module name, press 2 2
What is the module name? AK24_U17_FAULT
```

The V option returns the user to the display of the view menu.

B.4.4 Search

Selection of menu item 4 searches the database for a specified field entry and displays to the screen all records that match the search criteria. Note that the term that is searched must match exactly the contents of the specified field. When the user is through with the search, a carriage return is entered to return to the View Menu.

Menu Item 4

```
What field are you searching on:
Name, Test, Parameter, Units? name
What term are you searching for? (Please place in quotes) "alj634"
```

B.4.5 Addition

Selection of menu item 5 results in a display similar to the main menu display when creating a database. The user provides the contents of each field and must confirm the accuracy of the record. When the information is confirmed the record is appended to the existing database and the user is returned to the main menu. For repeated addition, the user must repeat this procedure. The following example demonstrates the addition of two records to a test database.

Select menu option 5 from the View Menu:

```
What is the name? ala67b
What is the test point? a78j5
What is the parameter? volts
What are the units? volts
```

```
What are the qualifying values? (list ok bad, etc) ok bad
What is the min (Press <Enter> if none)? -1.9
What is the max (Press <Enter> if none)? 1.9
What is the cost?
What are the prerequisites (Put a space between each)?
What are the instructions? (Press <Enter> if none)?
What is the type (Press <Enter> if none)? D
What is the focal module?

Name a1a67b
Test Point a78j5
Parameter volts
Units volts
Qualifying Values ok bad Min -1.9    Max 1.9
Cost
Prerequisites
Instruction
Type D
Focal Module

Is this correct (Y/N)? y
```

Return to View Menu and reselect menu option 5.

```
What is the name? A1A78J5
What is the test point? a78j6
What is the parameter? power
What are the units?
What are the qualifying values? (list ok bad, etc) ok hi lo
What is the min (Press <Enter> if none)? -25.25
What is the max (Press <Enter> if none)? 25.25
What is the cost?
What are the prerequisites (Put a space between each)?
What are the instructions? (Press <Enter> if none)?
What is the type (Press <Enter> if none)? D
What is the focal module?

Name A1A78J5
Test Point a78j6
Parameter power
Units
Qualifying Values ok hi lo Min -25.25    Max 25.25
Cost
Prerequisites
Instruction
Type D
Focal Module

Is this correct (Y/N)? y
```

B.4.6 Return to Main Menu

Selection of menu item **6** returns the user to the main menu.

APPENDIX C: FISDM.C PROGRAM LISTING

Listed below is the source code for *fisdm.c*. The contents of the included file, *fisdm.h*, is listed in Appendix A. The main functions are described in the above documentation.

```
/*

            The Fault Isolation System Database Manager (FISDM)
     Developed by Code 5524, Naval Research Laboratory, Washington DC 20375

   Author:  Chris Barclay
   Date:    October 25, 1991
   Object:  This program is to simplify the reading, modifying and general
            upkeep of the test and rule databases.
   Name:    fisdm.c (FIS Database Manager)
   Version: works on Sun OS 4.1 and 4.1.1, Empress 4.5

   barklay@itd.nrl.navy.mil
*/

#include "fisdm.h"

#define  LINE_LENGTH    300
#define      HEADING    "NAME,TEST   POINT,PARAMETER,UNITS,QUAL
VAL,MIN,MAX,COST,PREREQUISITES,COST,INSTRUCTION,TEXT,TYPE"  /*this is used by
panorama*/
#define  HEADING2 "MODULE, FRATE, CAUSE, EFFECT, TYPE, PRECONDITION"

/***************** DATA STRUCTURES ******************/
struct
{
  char info[30];
} module_base[300];    /*this keeps track of the modules for data_to_fis*/

struct tests
{
  char name[20], test_point[20], parameter[30], units[25],
       qual_val1[20], min[8], max[8], cost[50], prereq[100],
       instruction[30], test[30], type[30];
} testvar;                  /*test variable structure*/

struct rules
{
  char module[40], frate[6], cause[50], effect[50], type[30], precond[30];
} rulevar;                  /*rule variable structure*/

struct indexing
{
  char name[30];
} indexed[10];

struct stats     /*this is used to find out information about files*/
```

```
{
  dev_t     st_dev;
  ino_t     st_ino;
  u_short   st_mode;
  short     st_nlink;
  short     st_uid;
  short     st_gid;
  dev_t     st_rdev;
  off_t     st_size;
  time_t    st_atime;
  int       st_spare1;
  time_t    st_mtime;
  int       st_spare2;
  time_t    st_ctime;
  int       st_spare3;
  long      st_blksize;
  long      st_blocks;
  long      st_spare4[2];
} buf;

/*******************************************************/

/****************** VARIABLES ***********************/
int aindex, ch, comma, costlo, costhi, counter2, i, ii,
    instruclo, instruchi, menu_ok, minlo, minhi, maxlo, maxhi,
    namelo, namehi, paramlo, paramhi, prereqlo, prereqhi,
    previous, quallo, qualhi, startup, textlo, texthi,
    tpointlo, tpointhi, typelo, typehi, unithi, unitlo, wordcount;

char rend, response[30], answer1[5], datafilename[30], STRING[95],
     answer, a, ord[100], response2[30], bindex[3], info[LINE_LENGTH],
     *car(), *cdr();

FILE *fopen(), *fp1;

/*******************************************************/

msmain()
{
  aindex=0;       /*this tells the computer amount of fields indexed*/
  mxerret=1;      /*This sets the error mechanism*/
  menu_ok=1;      /*This begins the main menu*/
  startup=0;
  /*system("setenv MSVALSEP  "); this changes rules_stand params*/
  while (menu_ok==1)
  {
    if (startup==0)
      startup=10;
     else
      a=getchar();
    system("clear");
    printf("\n");
    printf("            ~~~~~~~~~~~~~~~  MAIN  MENU  ~~~~~~~~~~~~~~~\n");
    printf("            ~~~~~~~~~~~~~~~  ----------  ~~~~~~~~~~~~~~~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 1)   Conversion routines                 ~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 2)   Print database                      ~\n");
```

```
    printf("            ~                                              ~\n");
    printf("            ~ 3)   Display current databases               ~\n");
    printf("            ~                                              ~\n");
    printf("            ~ 4)   View database                           ~\n");
    printf("            ~                                              ~\n");
    printf("            ~ 5)   Create database                         ~\n");
    printf("            ~                                              ~\n");
    printf("            ~ 6)   Quit                                    ~\n");
    printf("            ~                                              ~\n");
    printf("            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
    printf("\n");
    printf("Input in the number, please ");
    answer=getchar();
    switch(answer)
    {
      case '1':   convert();
                  break;
      case '2':   pretty_print ();
                  break;
      case '3':   mscall(DATABASE, "display db");
                  printf("\nPress enter when ready.");
                  answer=getchar();
                  break;
      case '4':   view ();
                  break;
      case '5':   create ();
                  break;
      case '6':   menu_ok=0;
                  if (aindex>0)
                  for (i=1; i<=aindex; i++)  /*delete all existing indexes*/
                  {
                    strcpy(STRING, "drop index on ");
                    strcat(STRING, indexed[i].name);
                    mscall(DATABASE, STRING);
                  }
                  break;
      case 'v':   printf("Version for ITD2, Sun 3/80, October 25, 1991\n");
                  printf("Christopher Barclay, NRL Code 5524\n");
                  break;
      default:    printf("Incorrect response.\n");
                  break;
    }
  }
}

/************************* FUNCTIONS ***************************/

convert ()

{
  int imdone;

  for(;;)
  {
    system("clear");
    a=getchar();
    printf("\n");
    printf("            ~~~~~~~~~~~~  CONVERSION  MENU ~~~~~~~~~~~\n");
```

```
    printf("            ~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~ ~~~~~~~~~~~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 1)   Convert from comma to database      ~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 2)   Convert from database to comma      ~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 3)   Convert from FIS to database        ~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 4)   Convert from database to FIS        ~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 5)   Convert from text to database       ~\n");
    printf("            ~                                          ~\n");
    printf("            ~ 6)   Return to main menu                 ~\n");
    printf("            ~                                          ~\n");
    printf("            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
    printf("\n");
    printf("Input in the number, please ");
    answer=getchar();
    switch(answer)
    {
      case '1':     ascii_to_data ();
                    break;
      case '2':     data_to_ascii ();
                    break;
      case '3':     fis_to_data ();
                    break;
      case '4':     data_to_fis ();
                    break;
      case '5':     text_to_data ();
                    break;
      case '6':     imdone=20;
                    break;
      default:      printf("Incorrect response.\n");
                    break;
    }
  if(imdone==20)
    break;
  }
}

pretty_print ()

{
  int first_one, mindex, maybe, linenumber;
  char response[30], response2[10];

  ch=0;
  do
  {
    printf("What file do you want printed? ");
    scanf("%s", response);
    if (mxopen(DATABASE, response, "r"))
      ch=1;
    if (ch!=1)
      printf("That file doesn't exist\n");
  } while (ch != 1);
  printf("Is it rules or tests? ");
  scanf("%s", response2);
```

```
  do
  {
    printf("What do you want to name the print file? ");
    scanf("%s", datafilename);
    if (ch=access(datafilename, 0)==0)
    {
      printf("The file already exists.  Overwrite Y/N? ");
      scanf("%s", answer1);
    }
    if ((ch==1 && (answer1[0]=='y' || answer1[0]=='Y')) || ch!=1)
      fp1=fopen(datafilename, "w");
  } while (ch==1 && (answer1[0]=='n' || answer1[0]=='N'));

  if (response2[0]=='r' || response2[0]=='R')
  {
    mirdex=0;
       /*now, get all the names in the rules database together*/
    mxgetbegin(response, CHARNIL);
    while (mxget())
    {
      strcpy(rulevar.module, mxgetvs("module"));
      maybe=20;
      for (ii=0; ii<=mindex; ii++)
        if (strcmp(rulevar.module, module_base[ii].info) == 0)
          maybe=0;
        if (maybe != 0)
          strcpy(module_base[mindex++].info, rulevar.module);
    }    /*add to database if it's not there*/
    mxgetend ();
    for (ii=0; ii<mindex; ++ii)
    {
      first_one=0;
      linenumber=0;
      mxqcon("=", "module", module_base[ii].info);
      mxgetbegin(response, CHARNIL);
      while (mxget(;)
      {
        strcpy(rulevar.module, mxgetvs("module"));
        strcpy(rulevar.cause, mxgetvs("cause"));
        strcpy(rulevar.effect, mxgetvs("effect"));
        strcpy(rulevar.type, mxgetvs("type"));
        strcpy(rulevar.precond, mxgetvs("precond"));
        if(first_one==0)
        {
          fprintf(fp1, "\n Module: %s\n", rulevar.module);
           fprintf(fp1, "\nNo          Cause                          Effect
Type\n\n");
          first_one=10;
        }
        ++linenumber;
           fprintf(fp1, "%d     %s         %s         %s\n", linenumber,
rulevar.cause,
                rulevar.effect, rulevar.type);
      }
      mxgetend();
    }
    mxclose(response);
  }
```

```
  if (response2[0]=='T' || response2[0]=='t')
  {
                                        fprintf(fp1,          "
$3\n");
        fprintf(fp1,  "                                                $1
POSS\n");
      fprintf(fp1, "                                    TEST            $2
QUAL  $4    $5      COST\n");
    fprintf(fp1, "  NAME                POINT    PARAMETER          UNITS
RES      MIN      MAX        SECONDS          PREREQUISITES          NAME
PARAMETER\n\n\n");
    mxgetbegin(response, CHARNIL);
    while (mxget ())
    {
      strcpy(testvar.name, mxgetvs("name"));
      strcpy(testvar.test_point, mxgetvs("test_point"));
      strcpy(testvar.parameter, mxgetvs("parameter"));
      strcpy(testvar.units, mxgetvs("units"));
      strcpy(testvar.qual_val1, mxgetvs("qual1"));
      strcpy(testvar.min, mxgetvs("min"));
      strcpy(testvar.max, mxgetvs("max"));
      strcpy(testvar.cost, mxgetvs("cost"));
      strcpy(testvar.prereq, mxgetvs("prereq"));
      strcpy(testvar.instruction, mxgetvs("instruction"));
      strcpy(testvar.test, mxgetvs("test"));
      strcpy(testvar.type, mxgetvs("type"));
       if (strlen(testvar.name)<15)  /*if it's not long enough, add some
spaces*/        for(counter2=strlen(testvar.name); counter2<15; ++counter2)
          testvar.name[counter2]=' ';
      if (strlen(testvar.test_point)<11)
        for (counter2=strlen(testvar.test_point); counter2<11; ++counter2)
          testvar.test_point[counter2]=' ';
      if (strlen(testvar.parameter)<17)
        for (counter2=strlen(testvar.parameter); counter2<17; ++counter2)
          testvar.parameter[counter2]=' ';
      if (strlen(testvar.units)<13)
        for (counter2=strlen(testvar.units); counter2<13; ++counter2)
          testvar.units[counter2]=' ';
       /*printing section*/
      fprintf(fp1, "%s %s %s %s", testvar.name, testvar.test_point,
              testvar. parameter, testvar.units);
      fprintf(fp1, " %s    %s  %s     %s\n", testvar.qual_val1, testvar.min,
              testvar.max, testvar.cost);
      fprintf(fp1, "%s   %s\n", testvar.prereq, testvar.instruction);
    }
    mxgetend();
    mxclose(response);
  }
  fclose(fp1);
}

text_to_data ()

{
  int space, length, lineno;
  char string[150], *temp;
```

```
  strcpy (testvar.type, "D");  /*temp until it receives value*/

  printf("Are you working with tests or rules? ");
  scanf("%s", response);

  do
  {
    printf ("Enter the %s file name. ", response);
    scanf ("%s", response2);
    if ((fp1=fopen(response2, "r")) == NULL)
      printf("The file doesn't exist.\n");
  } while (fp1==NULL);

  do
  {
    printf("What would you like the database file to be named? ");
    scanf("%s", response2);
    if (mxopen(DATABASE, response2, "u"))
    {
      printf("Database already exists.\n");
      ch=1;
      mxclose(response2);
    }
     else
      if (response[0]=='r' || response[0]=='R')
      {
        strcpy(STRING, "create ");
        strcat(STRING, response2);
        strcat(STRING, " from rules_stand;");
                       /*this calls the file rules_stand from EMPRESS that
                         copies the data table information to the new file*/
        mscall(DATABASE, STRING);
        ch=0;
        mxopen(DATABASE, response2, "u");
      }
       else
      {
        strcpy(STRING, "create ");
        strcat(STRING, response2);
        strcat(STRING, " from test_stand;");
        mscall(DATABASE, STRING);
        ch=0;
        mxopen(DATABASE, response2, "u");
      }
  } while (ch==1);

  if (response[0]=='T' || response[0]=='t')  /*then it's the tests*/
  {
    /* read any top blank lines */
    lineno=0;
    while (blank_line (fgets (string, 150, fp1)));
    fgets (string, 150, fp1);  /* read the first three lines*/
    fgets (string, 150, fp1);
     fgets (string, 150, fp1); /*read column heading to find the colum
position*/
    assign_headings(string);
    while (fgets (string, 150, fp1) != NULL)
      if (its_a_testline (string))
```

```
    {
      ++lineno;   /*thanks to c.sarteschi for help on this function*/
      assign_to_values (string);
      while ((fgets (string, 150, fp1) != NULL) && (!blank_line (string)))
        assign_to_values (string);
      mxputvs("name", testvar.name);
      mxputvs("test_point", testvar.test_point);
      mxputvs("parameter", testvar.parameter);
      mxputvs("units", testvar.units);
      mxputvs("qual1", testvar.qual_val1);
      mxputvs("min", testvar.min);
      mxputvs("max", testvar.max);
      mxputvs("cost", testvar.cost);
      mxputvs("prereq", testvar.prereq);
      mxputvs("instruction", testvar.instruction);
      mxputvs("test", testvar.test);
      /*mxputvs("type", testvar.type);*/
      mxadd(response2);
      if (!strcmp (testvar.name, ""))
        printf("Error in name, line number %d\n", lineno);
      if (!strcmp (testvar.test_point, ""))
        printf("Error in test point, line number %d\n", lineno);
      if (!strcmp (testvar.parameter, ""))
        printf("Error in parameter, line number %d\n", lineno);
      if (!strcmp (testvar.units, ""))
        printf("Error in units, line number %d\n", lineno);
      if (!strcmp (testvar.qual_val1, ""))
        printf("Error in qualifying value, line number %d\n", lineno);
      if (!strcmp (testvar.cost, ""))
        printf("Error in cost, line number %d\n", lineno);
      if (!strcmp (testvar.prereq, ""))
        printf("Error in prerequisites, line number %d\n", lineno);
      if (!strcmp (testvar.instruction, ""))
        printf("Error in instructions, line number %d\n", lineno);
      strcpy (testvar.name, "");
      strcpy (testvar.test_point, "");
      strcpy (testvar.parameter, "");
      strcpy (testvar.units, "");
      strcpy (testvar.qual_val1, "");
      strcpy (testvar.min, "");
      strcpy (testvar.max, "");
      strcpy (testvar.cost, "");
      strcpy (testvar.prereq, "");
      strcpy (testvar.instruction, "");
      strcpy (testvar.test, "");
      /* this is to be commented out until it is included in input file */
      /* strcpy (testvar.type, ""); */
    }
}

temp=malloc(LINE_LENGTH+1);
if (response[0]=='R' || response[0]=='r') /*then it's the rules*/
if (answer=fgets(info, LINE_LENGTH, fp1) != NULL)
for(;;)
{
  if ((temp=strpbrk(info, "W")) != NULL)
    if (strncmp (temp, "WORKING", 7) == 0)  /*skip 2 lines if W is found*/
    {
```

```
        if (answer=fgets(info, LINE_LENGTH, fp1) == NULL)
          break;
        if (answer=fgets(info, LINE_LENGTH, fp1) == NULL)
          break;
      }
    if (((temp=strpbrk(info, "M")) != NULL) &&
          (strncmp (temp, "Modu", 4) ==0))   /*look for the letter M*/
    {
      strcpy(rulevar.module, "");
      for (ii=8; temp[ii] != '\n'; ++ ii)
        rulevar.module[ii-8]=temp[ii];
      if (answer=fgets(info, LINE_LENGTH, fp1) == NULL)
        break;
      if (answer=fgets(info, LINE_LENGTH, fp1) == NULL)
        break;
      if (answer=fgets(info, LINE_LENGTH, fp1) == NULL)
        break;
      /*i=i+3;  here, just read in 3 new lines of data*/
    }
     else
    {
      ii=0;
      if ((length=strlen(info)) > 15)
       if (((strpbrk(info, "[")==NULL) && (info[ii]!='N')) ||
                 (((temp=strpbrk(info,  "[")) !=NULL)  &&  (strlen(info)-
strlen(temp)>5)))
      {                  /*is the line [deleted] ? */
        for(i=0; i<49; i++)
          rulevar.cause[i]=0x20;
        for(i=0; i<49; i++)
          rulevar.effect[i]=0x20;
        for(i=0; i<9; i++)
          rulevar.precond[i]=0x20;
        rulevar.cause[49]='\0';
        rulevar.effect[49]='\0';
        rulevar.precond[9]='\0';
        ii=85;
        for (counter2=85; info[counter2] != '\n'; ++counter2)
          if (info[counter2] != ' ')
            rulevar.precond[counter2-ii]=info[counter2];
           else
            ++ii;
        space=0;
        for (counter2=3; counter2<=length && space != 5; ++counter2)
          if ((rulevar.cause[counter2-3]=info[counter2]) == ' ')
            ++space;
        space=0;
        for (counter2=40; counter2<=length && space != 4; ++counter2)
          if ((rulevar.effect[counter2-40]=info[counter2]) == ' ')
            ++space;
        mxputvs("module", rulevar.module);
        mxputvs("cause", rulevar.cause);
        mxputvs("effect", rulevar.effect);
        mxputvs("type", rulevar.type);
        mxputvs("precond", rulevar.precond);
        mxadd(response2);
      }
    }
```

```
        /*clean up the variables*/
      strcpy(rulevar.cause, "");
      strcpy(rulevar.effect, "");
      strcpy(rulevar.precond, "");
      if (answer=fgets(info, LINE_LENGTH, fp1) == NULL)
        break;
    }
    mxclose(response2);
    fclose(fp1);
    free(temp);
}

assign_to_values (string)

char *string;

{
    int i, hi;
    char ord[50];
    int get_word();

    i = 0;
    while ((string [i] != '\n') && (string[i] != '\0'))
    {
      strcpy (ord, ""); /* clear word */
      /* find the first non-blank character */
      while (string [i] == ' ')
        ++i;

      if (get_word (string, i, &hi, ord))
        assign_word (ord, i, hi);
      i = hi + 1;
    }
}

int get_word (string, lo, hi, ord)

char *string;
int lo;
int *hi;
char *ord;

{
    int i;
    strcpy (ord, ""); /* intialize word */

    /*check for comments '[', '(', or asterisk and read past*/
    if (string [lo] == '[')
    {
      while (string [lo] != ']')
        ++lo;
      *hi = lo;
      return 0;  /* a comment not a real word */
    }

    if (string [lo] == '(')
    {
      while (string [lo] != ')')
```

```
      ++lo;
    *hi = lo;
    return 0;  /* a comment not a real word */
  }

  if (string [lo] == '*') /* comments out the rest of the line */
  {
    while (string [lo] != '\n')  /* read to the end of comment line */
      ++lo;
    *hi = lo - 1;
    return 0;  /* a comment not a real word */
   }

  i = 0;
  while ((string [lo] != ' ') && (string [lo] != '\0') && (string [lo] !=
'\n'))
    ord [i++] = string[lo++];
  ord[i] = '\0';
  *hi = lo - 1;
  return 1;
}

int its_a_testline (string)

char *string;

{
  if (isalpha (string[0]))
    return 1;
   else
    return 0;
}

assign_word (ord, lo, hi)

char *ord;
int lo;
int hi;

{
  int i;

  for (i=lo; i<=hi; ++i)
  {
    if ((i>=namelo) && (i<=namehi))
    {
      strcpy (testvar.name, ord);
      break;
    }
    if ((i>=tpointlo) && (i<=tpointhi))
    {
      strcpy (testvar.test_point, ord);
      break;
    }
    if ((i>=paramlo) && (i<=paramhi))
    {
      strcpy (testvar.parameter, ord);
      break;
```

```
}
if ((i>=unitlo) && (i<=unithi))
{
  strcpy (testvar.units, ord);
  break;
}
if ((i>=quallo) && (i<=qualhi))
{
  if (strcmp (testvar.qual_val1, "")) /*not the same returns diff */
  {
    strcat (testvar.qual_val1, " ");
    strcat (testvar.qual_val1, ord);
  }
   else
    strcpy (testvar.qual_val1, ord);
  break;
}

if ((i>=minlo) && (i<=minhi))
{
  if (strcmp (testvar.min, "")) /*not the same returns diff */
  {
    strcat (testvar.min, " ");
    strcat (testvar.min, ord);
  }
   else
    strcpy (testvar.min, ord);
  break;
}

if ((i>=maxlo) && (i<=maxhi))
{
  if (strcmp (testvar.max, "")) /*not the same returns diff */
  {
    strcat (testvar.max, " ");
    strcat (testvar.max, ord);
  }
   else
    strcpy (testvar.max, ord);
  break;
}

if ((i>=costlo) && (i<=costhi))
{
  strcpy (testvar.cost, ord);
  break;
}

if ((i>=prereqlo) && (i<=prereqhi))
{
  if (strcmp (testvar.prereq, "")) /*not the same returns diff */
  {
    strcat (testvar.prereq, " ");
    strcat (testvar.prereq, ord);
  }
   else
    strcpy (testvar.prereq, ord);
  break;
```

```
    }

    if ((i>=instruclo) && (i<=instruchi))
    {
      strcpy (testvar.instruction, ord);
      break;
    }

    if ((i>=textlo) && (i<=texthi))
    {
      if (strcmp (testvar.test, "")) /*not the same returns diff */
      {
        strcat (testvar.test, " ");
        strcat (testvar.test, ord);
      }
       else
        strcpy (testvar.test, ord);
      break;
    }
    /* ADD ABOUT TYPES */
  }
}


int find_begin_word (position, string)

int *position;
char *string;

{
  while (string[*position] == ' ')
    ++ (*position);
  return (*position);
}

int find_end_word (position, string)

int *position;
char *string;

{
  while ((string[*position] != ' ') && (string[*position] != '\0'))
    ++ (*position);
  return (*position - 1);
}

assign_headings(string)

char *string;

{
  int count;
  count = 0;

  namelo = find_begin_word (&count, string);
  namehi = find_end_word (&count, string);

  tpointlo = find_begin_word (&count, string);
```

```
  tpointhi = find_end_word (&count, string);

  paramlo = find_begin_word (&count, string);
  paramhi = find_end_word (&count, string);

  unitlo = find_begin_word (&count, string);
  unithi = find_end_word (&count, string);

  quallo = find_begin_word (&count, string);
  qualhi = find_end_word (&count, string);

  minlo = find_begin_word (&count, string);
  minhi = find_end_word (&count, string);

  maxlo = find_begin_word (&count, string);
  maxhi = find_end_word (&count, string);

  costlo = find_begin_word (&count, string);
  costhi = find_end_word (&count, string);

  prereqlo = find_begin_word (&count, string);
  prereqhi = find_end_word (&count, string);

  instruclo = find_begin_word (&count, string);
  instruchi = find_end_word (&count, string);

  textlo = find_begin_word (&count, string);
  texthi = find_end_word (&count, string);

  /* not yet in the file, when it is need to add the && (string[count++] !=
'\0')) */
  /*while (string[count++] == ' ');
  typelo = find_begin_word (&count, string);
  typehi = find_end_word (&count, string);
  */
}

int blank_line (string)

char *string;

{
  int length, i;

  if (string[0] == '\n')
    return 1; /* it is a blank line */

  length = strlen (string);
  for (i=0; i<(length-1); ++i)
    if (string[i] != ' ')
      return 0; /* it is not a blank line */

  return 1;  /* then it is a blank line */
}

fis_to_data ()

{
```

```
char y[75000], z[30000], z2[8000], ok_bad[150], answer, *temp, *string2,
     *string3, new[200], new2[500], *xstr;

int i, j, zzz, rule_done, right_paren, left_paren, ucount, ui;

struct tm *ltime;

time_t lt;

do
{
  printf("What is the name of your FIS or GUI produced file? ");
  scanf("%s", datafilename);
  if ((fp1=fopen(datafilename, "r")) == NULL)
    printf("File doesn't exist.\n");
} while (fp1==NULL);
fclose(fp1);  /*close, make sure it's been folded, then reopen*/

do
{
  printf("What do you want to name the new test file? ");
  scanf("%s", response);
  if (mxopen(DATABASE, response, "u"))
  {
    ch=1;
    printf("That file already exists.  Pick another.\n");
    mxclose(response);
  }
   else
  {
    strcpy(STRING, "create ");
    strcat(STRING, response);
    strcat(STRING, " from test_stand");
    ch=0;
    mscall(DATABASE, STRING);
  }
} while (ch != 0);

do
{
  printf("What do you want to name the new rule file? ");
  scanf("%s", response2);
  if (mxopen(DATABASE, response2, "u"))
  {
    ch=1;
    printf("That file already exists.  Pick another.\n");
    mxclose(response2);
  }
   else
  {
    strcpy(STRING, "create ");
    strcat(STRING, response2);
    strcat(STRING, " from rules_stand");
    ch=0;
    mscall(DATABASE, STRING);
  }
} while (ch != 0);
printf("\nPlease wait...\n");
```

```
  j=0;
  left_paren=0;
  right_paren=0;
  rule_done=0;
  ucount=0;
  lt=time(NULL);
  ltime=localtime(&lt);
  xstr=asctime(ltime);
  for(ui=0; ui<strlen(xstr); ++ui)
    if(xstr[ui]!=' ' && xstr[ui]!='\0' && xstr[ui]!='\n')
      new[ucount++]=xstr[ui];
  strcpy(new2, "fold -80 "); /*make sure it's only 80 characters across*/
  strcat(new2, datafilename);
  strcat(new2, ">");
  strcat(new2, new);
  system(new2);
  strcpy(new2, "rm ");
  strcat(new2, datafilename);
  system(new2);
  strcpy(new2, "mv ");
  strcat(new2, new);
  strcat(new2, " ");
  strcat(new2, datafilename);
  system(new2);
  fp1=fopen(datafilename, "r");
  stat(datafilename, &buf);  /*this gets info about the file - like size*/
  if((string2=(char *)malloc(buf.st_size))==NULL)
    exit(0); /*this makes sure to malloc (if available) the  */
  if((string3=(char *)malloc(buf.st_size))==NULL) /*correct amount of memory
*/
    exit(0); /*if not, program terminates */
  while ((answer=fgets(info, LINE_LENGTH, fp1) != NULL)
           &&  rule_done !=50) /*read in line-by-line until done with tests*/
    if (info[0] != 'N' && info[1] != 'I')
      for (i=0; i<=strlen(info); ++i)
        if ((info[i] != '\n') && (info[i] != '\0'))  /*put in long string if*/
          if (info[i]=='N' && info[i+1]=='I' && info[i+2]=='L') /*no newline*/
          {
            if (rule_done != 40)
            {
              string3[j++]='(';   /*converts nil to ()*/
              string3[j++]=')';
              i=i+2;
            }
             else
            {
              string2[j++]='(';
              string2[j++]=')';
              i=i+2;
            }
          }
           else
          {
            if (info[i]=='(')
              ++left_paren;
            if (info[i]==')')
              ++right_paren;
```

```
            if (rule_done != 40)
              string3[j++]=info[i];  /*read in rules*/
             else
              string2[j++]=info[i];  /*read in tests*/
            if((left_paren == right_paren) && (left_paren>1))
            {
              temp=strpbrk(string3, "N");
              if (strncmp(temp, "NAME", 4)==0) /*is it graphics*/
              {
                j=0; /*it's not graphics*/
                if(rule_done==40)
                {
                  rule_done=50;
                  break;
                }
                 else
                  rule_done=40;/*once the rules have been read in, move to
tests*/
              }
               else
                j=0;
            }
          }
  fclose(fp1);
  mxopen(DATABASE, response2, "u");
  do       /*get new module*/
  {
    strcpy(y, car(string3));                    /*this is where the info is*/
    strcpy(z, car(y));  /*cadar y is the module name*/
    strcpy(z, cdr(z));
    strcpy(z, car(z));
    strcpy(rulevar.module, z);
    strcpy(z, cdr(y));
    strcpy(z2, car(z));
    strcpy(z2, cdr(z2));
    strcpy(z2, car(z2));
    strcpy(rulevar.frate, z2);
    strcpy(z, cdr(z));   /*caddr y gives causal rules*/
    strcpy(z, car(z));  /*now find out the info within*/
    strcpy(z, cdr(z)); /*caadr caddr z is where the first one is found*/
    strcpy(z, car(z));
    do     /*get the info out of the module*/
    {
      strcpy(z2, car(z));
      strcpy(rulevar.precond, car(z2)); /*car caadr caddr*/
      strcpy(z2, cdr(z2));
      strcpy(rulevar.cause, car(z2));   /*cadr caadr caddr*/
      strcpy(z2, cdr(z2));
      strcpy(rulevar.effect, car(z2));  /*caddr caadr caddr*/
      if (rulevar.module[0]=='(')
      {
        for (i=0; rulevar.module[i+1] != ')'; i++)
          rulevar.module[i]=rulevar.module[i+1];
        rulevar.module[i]='\0';
      }
      if (rulevar.cause[0]=='(')
      {
        for (i=0; rulevar.cause[i+1] != ')'; i++)
```

```
        rulevar.cause[i]=rulevar.cause[i+1];
      rulevar.cause[i]='\0';
    }
    if (rulevar.effect[0]=='(')
    {
      for (i=0; rulevar.effect[i+1] != ')'; i++)
        rulevar.effect[i]=rulevar.effect[i+1];
      rulevar.effect[i]='\0';
    }
    if (rulevar.precond[0]=='(')
    {
      for (i=0; rulevar.precond[i+1] != ')'; i++)
        rulevar.precond[i]=rulevar.precond[i+1];
      rulevar.precond[i]='\0';
    }
    mxputvs("module", rulevar.module);
    if(what_line (rulevar.frate))
      mxputvs("frate", rulevar.frate);
     else
      mxputvs("frate", "0.1"); /*sets default value if no data*/
    mxputvs("cause", rulevar.cause);
    mxputvs("effect", rulevar.effect);
    if(what_line (rulevar.precond))
      mxputvs("precond", rulevar.precond);
     else
      mxputvs("precond", "t");
    mxadd(response2);
    strcpy(z, cdr(z));   /*this gets rid of what was just worked on*/
  } while (strncmp (z, "()", 2) != 0);
  strcpy(string3, cdr(string3));  /*this moves past the first one*/
} while (strncmp (string3, "()", 2) != 0); /*end of rules when ()*/
mxclose(response2);
free(string3);

mxopen(DATABASE, response, "u");
do
{
  strcpy(y, car(string2));
  strcpy(testvar.test_point, car(y));
  strcpy(y, cdr(y));
  do
  {
    for (zzz=0; zzz<7; zzz++)
      testvar.min[zzz]= 0x20;
    for (zzz=0; zzz<7; zzz++)
      testvar.max[zzz]= 0x20;
    testvar.min[7]='\0';
    testvar.max[7]='\0';
    strcpy(z, car(y));
    strcpy(testvar.name, car(z));
    strcpy(z, cdr(z));
    strcpy(testvar.parameter, car(z));
    strcpy(z, cdr(z));
    strcpy(z, cdr(z));
    strcpy(ok_bad, car(z));  /*need to get qual vals out of here*/
    strcpy(z2, car(ok_bad));
    strcpy(z2, cdr(z2));
    if (!strcmp (z2, "()"))  /*there's no min, max values*/
```

```
{
  j=0;
  for (i=0; i<=strlen(ok_bad); ++i)
    if(ok_bad[i]!='(' && ok_bad[i]!=')')
      testvar.qual_val1[j++]=ok_bad[i];
}
 else
{
  strcpy(z2, car(ok_bad));
  strcpy(z2, cdr(z2));
  strcpy(z2, car(z2));
  if (z2[1]=='(')
    strcpy(z2, car(z2));
  strcpy(testvar.min, car(z2));
  strcpy(z2, cdr(z2));
  strcpy(testvar.max, car(z2));
  strcpy(z2, car(ok_bad));
  strcpy(testvar.qual_val1, car(z2)); /*starting to fill up qual vals*/
  strcpy(ok_bad, cdr(ok_bad));
  while (strcmp (ok_bad, "()"))
  {
    strcpy(z2, car(ok_bad));
    strcat(testvar.qual_val1, " ");
    strcat(testvar.qual_val1, car(z2));
    strcpy(ok_bad, cdr(ok_bad));
  }
}

strcpy(z, cdr(z));
strcpy(testvar.units, car(z));
strcpy(z, cdr(z));
strcpy(testvar.test, car(z));
strcpy(z, cdr(z));
strcpy(testvar.cost, car(z));
strcpy(z, cdr(z));
strcpy(testvar.type, car(z));  /*this is the focal module, it's NIL*/
if (testvar.name[0]=='(')      /*if undefined*/
{
  for (i=0; testvar.name[i+1] != ')'; i++)
    testvar.name[i]=testvar.name[i+1];
  testvar.name[i]='\0';
}
if (testvar.test_point[0]=='(')
{
  for (i=0; testvar.test_point[i+1] != ')'; i++)
    testvar.test_point[i]=testvar.test_point[i+1];
  testvar.test_point[i]='\0';
}
if (testvar.parameter[0]=='(')
{
  for (i=0; testvar.parameter[i+1] != ')'; i++)
    testvar.parameter[i]=testvar.parameter[i+1];
  testvar.parameter[i]='\0';
}
if (testvar.qual_val1[0]=='(')
{
  for (i=0; testvar.qual_val1[i+1] != ')'; i++)
    testvar.qual_val1[i]=testvar.qual_val1[i+1];
```

```
        testvar.qual_val1[i]='\0';
      }
      if (testvar.units[0]=='(')
      {
        for (i=0; testvar.units[i+1] != ')'; i++)
          testvar.units[i]=testvar.units[i+1];
        testvar.units[i]='\0';
      }
      if (testvar.type[0]=='(')
      {
        for(i=0; testvar.type[i+1] !=')'; i++)
          testvar.type[i]=testvar.type[i+1];
        testvar.type[i]='\0';
      }
      mxputvs("name", testvar.name);
      mxputvs("test_point", testvar.test_point);
      mxputvs("parameter", testvar.parameter);
      mxputvs("qual1", testvar.qual_val1);
      mxputvs("min", testvar.min);
      mxputvs("max", testvar.max);
      if(what_line (testvar.units))
        mxputvs("units", testvar.units);
       else
        mxputvs("units", "NIL");
      if(what_line (testvar.test))
        mxputvs("test", testvar.test);
       else
        mxputvs("test", "D");
      if(what_line (testvar.cost))
        mxputvs("cost", testvar.cost);
       else
        mxputvs("cost", "10");
      if(what_line (testvar.type))
        mxputvs("type", testvar.type);
       else
        mxputvs("type", "NIL");
      mxadd(response);
      strcpy(y, cdr(y));
      mxsetnull (response);
    } while(strncmp (y, "()", 2) != 0);
    strcpy(string2, cdr(string2));          /*next test*/
  } while (strncmp (string2, "()", 2) != 0); /*end of tests when ()*/
  mxclose(response);
  free(string2);
}
  /*to read string2, {while(*string2)  printf("%c", *string2++);}
  in sun lisp to read in power_temp.v - (setq data(open "power_temp.v"))
                                        (setq readin(read data)) for each line */

char *car (string)

char *string;

{
  int i, j, k, b, left_paren_count, right_paren_count;
  static char car_word[20000];

  i=0;
```

```
  j=0;
  left_paren_count = 0;
  right_paren_count = 0;
  car_word[0] = '\0';    /* clear static variable */
  /*make sure it's a list*/
  while (string[i] == ' ')   /* get rid of spaces */
    ++i;
  if (string[i] != '(' )  /* not a list - return error */
    return ("ERROR");
  /*it is a list - check to see if there's a nested car*/
  while (string[i+1] == ' ')   /* get rid of spaces inside first list*/
    ++i;
  if (string[i+1] != '(' ) /* no list */
  {
    ++i; /*move past the leading paren */
    while ((isalnum (string[i])) || (ispunct (string[i] ))
             && (string[i] != ')') && (string[i] != '('))/* a start of a word
*/
      car_word[j++] = string[i++];
  }
   else /* then we have a list, process differently */
  {
    k=i+1;
    /* find the correct depth of parens */
    for(;;)
    {
      if (string [k] == '(')
        ++left_paren_count;
      if (string [k] == ')')
        ++right_paren_count;
      if (right_paren_count == left_paren_count)
        break;
      ++k;
    }

    for (b=i+1; b<=k; ++b)
      car_word[j++] = string[b];
  }
  if (!strlen (car_word))
    strcpy(car_word, "()");   /* return empty set */
   else
    car_word[j] = '\0';

  return (car_word);
}

char *cdr (string)      /*start of cdr function*/

char *string;

{
  int i, j, k, right_paren_count, left_paren_count;
  static char cdr_word[350000];

  i=0;
  j=0;
  cdr_word[0]='\0';
  while (string[i] == ' ')   /*get rid of the spaces*/
```

```
    ++i;
  if (string[i] != '(')  /*if it's not a list, then error*/
    return ("ERROR");
  ++i;
  while (string[i] == ' ')
    ++i;
  if (string[i] != '(')  /*then advance to next paren or space*/
  {
    while ((string[i] != '(') && (string[i] != ')') && (string[i] != ' '))
      ++i;
    if (string[i]==')')
      strcpy(cdr_word, "()");  /*it's the empty set*/
     else
    {
      cdr_word[0]='(';
      if (string[i]==' ')
        ++i;
      k=strlen(string);
      for (j=i+1; j<=k; ++j)
        cdr_word[j-i]=string[j-1];
      cdr_word[j-i]='\0';
    }
  }
   else
  {
    left_paren_count=1;
    right_paren_count=0;
    ++i;
    while (left_paren_count != right_paren_count)
    {
      if (string[i]=='(')
        ++left_paren_count;
      if (string[i]==')')
        ++right_paren_count;
      ++i;
    }
    while (string[i]==' ')
      ++i;
    cdr_word[0]='(';
    k=strlen(string);
    for (j=i+1; j<=k; ++j)
      cdr_word[j-i]=string[j-1];
    cdr_word[j-i]='\0';
  }
  return (cdr_word);
}

data_to_fis ()

{
  int mindex, maybe, first_one, counts, d;

  char t1[10], t2[10], t3[10], qualvals[35];

  do
  {
    printf("What is the name of your rule database file? ");
    scanf("%s", response);
```

```
    if (mxopen(DATABASE, response, "r"))
      ch=1;
     else
    {
      ch=0;
      printf("Doesn't exist.\n");
    }
  } while (ch==0);
  mxclose(response);

  do
  {
    printf("What is the name of your test database file? ");
    scanf("%s", response2);
    if (mxopen(DATABASE, response2, "r"))
      ch=1;
     else
    {
      ch=0;
      printf("Doesn't exist.\n");
    }
  } while (ch==0);
  mxclose(response2);

  do
  {
    printf("What is the name of your new FIS file? ");
    scanf("%s", datafilename);
    strcat(datafilename, ".v");
    if (ch=access(datafilename, 0) == 0)
    {
      printf("The file already exists.  Overwrite Y/N?");
      scanf("%s", answer1);
    }
    if ((ch==1 && (answer1[0]=='y' || answer1[0]=='Y')) || ch != 1)
      fp1=fopen(datafilename, "w");
  } while (ch == 1 && (answer1[0]=='n' || answer1[0]=='N'));

  mindex=0;
  fprintf(fp1, "NIL\nNIL\nNIL\n(");      /*the start of the file*/
  mxopen(DATABASE, response, "r");    /*opening rules database*/
       /*now, get all the names in the rules database together*/
  mxgetbegin(response, CHARNIL);
  while (mxget())
  {
    strcpy(rulevar.module, mxgetvs("module"));
    maybe=20;
    for (ii=0; ii<=mindex; ii++)
      if (strcmp(rulevar.module, module_base[ii].info) == 0)
        maybe=0;
      if (maybe != 0)
        strcpy(module_base[mindex++].info, rulevar.module);
  }     /*add to database if it's not there*/
  mxgetend ();
  for (ii=0; ii<mindex; ++ii)
  {
     first_one=0;          /*this is to check for the 1st one, which is
different*/
```

```
    mxqcon("=", "module", module_base[ii].info);  /*see p34*/
    mxgetbegin(response, CHARNIL);
    while (mxget())
    {
      strcpy(rulevar.module, "");
      strcpy(rulevar.frate, "");
      strcpy(rulevar.cause, "");
      strcpy(rulevar.effect, "");
      strcpy(rulevar.precond, "");
      strcpy(rulevar.module, mxgetvs("module"));
      strcpy(rulevar.frate, mxgetvs("frate"));
      strcpy(rulevar.cause, mxgetvs("cause"));
      strcpy(rulevar.effect, mxgetvs("effect"));
      strcpy(rulevar.type, mxgetvs("type"));
      strcpy(rulevar.precond, mxgetvs("precond"));
      if(first_one==0)
      {
            fprintf(fp1, " ((NAME %s) (FRATE %s)\n           (CAUSAL-
RULES\n",rulevar.module, rulevar.frate);
        if (strcmp(rulevar.module, rulevar.cause)!=0)
        {
          fprintf(fp1, "          (\n            (%s (%s)\n", rulevar.precond,
rulevar.cause);
          fprintf(fp1, "             (%s))\n", rulevar.effect);
        }
         else
        {
          fprintf(fp1, "          (\n            (%s %s\n", rulevar.precond,
rulevar.cause);

          fprintf(fp1, "             (%s))\n", rulevar.effect);
        }
        ++first_one;
      }
       else
        if (strcmp(rulevar.module, rulevar.cause)!=0)
        {
                    fprintf(fp1,  "                        (%s  (%s)\n",
rulevar.precond,rulevar.cause);
          fprintf(fp1, "             (%s))\n", rulevar.effect);
        }
         else       /*this is in case the cause is the same as the module*/
        {
             fprintf(fp1, "                (%s %s\n", rulevar.precond,
rulevar.cause);
          fprintf(fp1, "             (%s))\n", rulevar.effect);
        }
    }
    fprintf(fp1, " )))\n");
    mxgetend();
  }
  fprintf(fp1, ")\n(");
  mxclose(response);
  mindex=0;
  mxopen(DATABASE, response2, "r");         /*opening tests database*/
  mxgetbegin(response2, CHARNIL);
  while (mxget())
  {
```

```
    strcpy(testvar.test_point, mxgetvs("test_point"));
    maybe=20;
    for (ii=0; ii<=mindex; ii++)
      if (strcmp(testvar.test_point, module_base[ii].info)==0)
        maybe=0;
    if (maybe != 0)
      strcpy(module_base[mindex++].info, testvar.test_point);
  }
  mxgetend ();
  for(ii=0; ii<mindex; ++ii)
  {
    first_one=0;
    mxqcon("=", "test_point", module_base[ii].info);
    mxgetbegin(response2, CHARNIL);
    while (mxget())
    {
      strcpy(testvar.name, "");
      strcpy(testvar.test_point, "");
      strcpy(testvar.cost, "");
      strcpy(testvar.test, "");
      strcpy(testvar.parameter, "");
      strcpy(testvar.qual_val1, "");
      strcpy(testvar.min, "");
      strcpy(testvar.max, "");
      strcpy(testvar.units, "");
      strcpy(testvar.type, "");
      strcpy(testvar.name, mxgetvs("name"));
      strcpy(testvar.test_point, mxgetvs("test_point"));
      strcpy(testvar.parameter, mxgetvs("parameter"));
      strcpy(testvar.units, mxgetvs("units"));
      strcpy(testvar.min, mxgetvs("min"));
      strcpy(testvar.max, mxgetvs("max"));
      strcpy(testvar.cost, mxgetvs("cost"));
      strcpy(testvar.test, mxgetvs("test"));
      strcpy(testvar.qual_val1, mxgetvs("qual1"));
      strcpy(testvar.type, mxgetvs("type"));
      if(first_one==0)
      {
        ++first_one;
        fprintf(fp1, "    (%s (%s %s\n          S1\n", testvar.test_point,
                  testvar.name, testvar.parameter);
      }
       else
        fprintf(fp1, "        (%s %s\n          S1\n", testvar.name,
                  testvar.parameter);
      if (testvar.min[0]=='\0')
      {
         /*new section below*/
        d=2;
        strcpy(qualvals, "((");
        for(i=0; i<=strlen(testvar.qual_val1); ++i)
        {
          if (testvar.qual_val1[i]!=' ')
            qualvals[d++]=testvar.qual_val1[i];
           else
          {
            strcat(qualvals, ") (");
            d=d+3;
```

```
        }
      }
      strcat(qualvals, "))\0");
      fprintf(fp1, "          %s\n", qualvals);
      for (i=0; i<35; i++)
        qualvals[i]='\0';
    }
     else
    {
      counts=0;
      d=0;
      for(i=0; i<10; i++)
      {
        t1[i]='\0';        /*max of 3 values*/
        t2[i]='\0';  t3[i]='\0';
      }
      for(i=0; i<=strlen(testvar.qual_val1); ++i)
      {
        if (testvar.qual_val1[i]==' ')
        {
          d=0;
          ++counts;
        }
         else
        {
          if (counts==0)
            t1[d++]=testvar.qual_val1[i];
          if (counts==1)
            t2[d++]=testvar.qual_val1[i];
          if (counts==2)
            t3[d++]=testvar.qual_val1[i];
        }
      }
      if (counts==1)
        fprintf(fp1, "            ((%s (%s %s)) (%s (-inf %s) (%s inf)))\n",
              t1, testvar.min, testvar.max, t2, testvar.min, testvar.max);
       else
        fprintf(fp1, "            ((%s ((%s %s))) (%s ((-inf %s))) (%s ((%s
inf))))\n", t1, testvar.min, testvar.max, t2, testvar.min, t3, testvar.max);
      }
      fprintf(fp1, "              %s\n             %s\n             %s\n
%s)\n",
                  testvar.units, testvar.test, testvar.cost, testvar.type);
    }
    fprintf(fp1, "  )\n");
    mxgetend();
  }
  fprintf(fp1, ")\nNIL\nNIL\nNIL");
  mxclose(response2);
  fclose(fp1);
}

ascii_to_data ()

{
  printf("Is the file format rules or tests? ");
  scanf("%s", response);
  do
```

```
{
  printf("What is the name of your %s file? ", response);
  scanf("%s", response2);
  if ((fp1=fopen(response2, "r")) == NULL)
    printf("File doesn't exist.\n");
} while (fp1 == NULL);

do
{
  printf("What do you want to name the new file? ");
  scanf("%s", datafilename);
  if (mxopen(DATABASE, datafilename, "u"))
    {
       ch=1;
       printf("That file already exists.  Pick another.\n");
       mxclose(datafilename);
    }
     else
                /*here's where we create the format*/
    if (response[0]=='r' || response[0]=='R')
    {
      strcpy(STRING, "create ");
      strcat(STRING, datafilename);
      strcat(STRING, " from rules_stand;");
                     /*this calls the file rules_stand from EMPRESS that
                       copies the data table information to the new file*/
      mscall(DATABASE, STRING);
      ch=0;
      mxopen(DATABASE, datafilename, "u");
    }
     else
    {
      strcpy(STRING, "create ");
      strcat(STRING, datafilename);
      strcat(STRING, " from test_stand;");
      mscall(DATABASE, STRING);
      ch=0;
      mxopen(DATABASE, datafilename, "u");
    }
} while (ch != 0);

while (rend=fgets(info, LINE_LENGTH, fp1) != NULL)
{
  if (response[0]=='t' || response[0]=='T')
  {
    comma=0;
    wordcount=0;
    for (ii=0; info[ii] != '\n'; ++ii)  /*this parses through a */
      if ((info[ii]!=',') && (info[ii]!='\n'))    /*line of data*/
      {
        ord[wordcount]=info[ii];
        ++wordcount;
        if (info[ii+1]=='\n' && comma<11)
          printf("Data File Corrupted: Not enough fields.\n");
      }
       else
      {
        for (i=wordcount; i<=100; ++i)
```

```
  ord[i]='\0';
++ comma;
switch(comma)   /*keeps track of what field the program is reading*/
{
  case 1:     mxputvs("name", ord);
              wordcount=0;
              break;
  case 2:     mxputvs("test_point", ord);
              wordcount=0;
              break;
  case 3:     mxputvs("parameter", ord);
              wordcount=0;
              break;
  case 4:     mxputvs("units", ord);
              wordcount=0;
              break;
  case 5:     mxputvs("qual1", ord);
              wordcount=0;
              break;
  case 6:     if (ord[1] == '\0')
                mxputvs("min", "");
               else
                mxputvs("min", ord);
              wordcount=0;
              break;
  case 7:     if (ord[1] == '\0')
                mxputvs("max", "");
               else
                mxputvs("max", ord);
              wordcount=0;
              break;
  case 8:     if (what_line(ord))
                mxputvs("cost", ord);
               else
                mxputvs("cost", "10");
              wordcount=0;
              break;
  case 9:     mxputvs("prereq", ord);
              wordcount=0;
              break;
  case 10:    if (ord[2] == '\0')
                mxputvs("instruction", "");
               else
                mxputvs("instruction", ord);
              wordcount=0;
              break;
  case 11:    if (what_line(ord))
                mxputvs("test", ord);
               else
                mxputvs("test", "D");
              wordcount=0;
              break;
  case 12:    if (what_line(ord))
                mxputvs("type", ord);
               else
                mxputvs("type", "NIL");
              wordcount=0;
              break;
```

```
            default:    printf("Data file corrupted: Too Many Fields.\n");
                        break;
          }
        }
      }

      if (response[0]=='r' || response[0]=='R')
      {
        comma=0;
        wordcount=0;
        for (ii=0; info[ii] != '\n'; ++ii)  /*this parses through a */
          if (info[ii]!=',')                /*a line of data*/
          {
            ord[wordcount]=info[ii];
            ++wordcount;
            if (info[ii+1]=='\n' && comma<5)
              printf("Data File Corrupted: Not enough fields.\n");
          }
           else
          {
            for (i=wordcount; i<=100; ++i)
              ord[i]='\0';
            ++comma;
            switch (comma)
            {
              case 1:     mxputvs("module", ord);
                          wordcount=0;
                          break;
              case 2:     if (what_line(ord))
                            mxputvs("frate", ord);
                           else
                            mxputvs("frate", "0.1");
                          wordcount=0;
                          break;
              case 3:     mxputvs("cause", ord);
                          wordcount=0;
                          break;
              case 4:     mxputvs("effect", ord);
                          wordcount=0;
                          break;
              case 5:     mxputvs("type", ord);
                          wordcount=0;
                          break;
              case 6:     if (what_line(ord))
                            mxputvs("precond", ord);
                           else
                            mxputvs("precond", "t");
                          wordcount=0;
                          break;
              default:    printf("Data file corrupted: Too Many Fields.\n");
                          break;
            }
          }
        if (info[ii]=='\n' && ++comma==6)
          if (what_line(ord))
            mxputvs("precond", ord);
           else
            mxputvs("precond", "t");
```

```
      /*the above was added because with the \n, it didn't read the last
field*/
    }
    mxadd(datafilename);
    for (wordcount=0; wordcount<=LINE_LENGTH; ++wordcount)
      info[wordcount]=='\0';
    wordcount=0;
  }
  fclose(fp1);
  mxclose(datafilename);
}

what_line (data)

char data[100];

{
  int i, b;

  b=1;
   for (i=0; i<strlen(data) && b==1; ++i) /*this function determines if
there's*/
    if (isalnum(data[i]))   /*anything on a line...set a default if not*/
    {
      b=2;
      return 1;
    }
  return 0;
}


data_to_ascii ()

{
  printf("Is the file format rules or tests? ");
  scanf("%s", response);
  do
  {
    printf("What is the name of your database file? ");
    scanf("%s", datafilename);
    if(mxopen(DATABASE, datafilename, "r"))
      ch=1;
     else
    {
      ch=0;
      printf("Doesn't exist.\n");
    }
  } while (ch == 0);

  do
  {
    printf("What is the name of your new %s file? ", response);
    scanf("%s", response2);
    if (ch=access(response2, 0) == 0)
    {
      printf("The file already exists.  Overwrite Y/N?");
      scanf("%s", answer1);
    }
```

```
    if ((ch==1 && (answer1[0]=='y' || answer1[0]=='Y')) || ch != 1)
      fp1=fopen(response2, "w");
  } while (ch == 1 && (answer1[0]=='N' || answer1[0]=='n'));

  if (response[0]=='t' || response[0]=='T')
  {
    mxgetbegin(datafilename, CHARNIL);
    while (mxget ())
    {
      strcpy(testvar.name, mxgetvs("name"));
      strcpy(testvar.test_point, mxgetvs("test_point"));
      strcpy(testvar.parameter, mxgetvs("parameter"));
      strcpy(testvar.units, mxgetvs("units"));
      strcpy(testvar.qual_val1, mxgetvs("qual1"));
      strcpy(testvar.min, mxgetvs("min"));
      strcpy(testvar.max, mxgetvs("max"));
      strcpy(testvar.cost, mxgetvs("cost"));
      strcpy(testvar.prereq, mxgetvs("prereq"));
      strcpy(testvar.instruction, mxgetvs("instruction"));
      strcpy(testvar.test, mxgetvs("test"));
      strcpy(testvar.type, mxgetvs("type"));
      fprintf(fp1, "%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s\n",
             testvar.name, testvar.test_point, testvar.parameter,
             testvar.units, testvar.qual_val1, testvar.min, testvar.max,
             testvar.cost, testvar.prereq, testvar.instruction,
             testvar.test, testvar.type);
    }
    mxgetend ();
    mxclose(datafilename);
    fclose(fp1);
  }

  if (response[0]=='r' || response[0]=='R')
  {
    mxgetbegin(datafilename, CHARNIL);
    while (mxget ())
    {
      strcpy(rulevar.module, mxgetvs("module"));
      strcpy(rulevar.frate, mxgetvs("frate"));
      strcpy(rulevar.cause, mxgetvs("cause"));
      strcpy(rulevar.effect, mxgetvs("effect"));
      strcpy(rulevar.type, mxgetvs("type"));
      strcpy(rulevar.precond, mxgetvs("precond"));
      fprintf(fp1, "%s, %s, %s, %s, %s, %s\n", rulevar.module, rulevar.frate,
             rulevar.cause, rulevar.effect, rulevar.type, rulevar.precond);
    }
    mxgetend();
    mxclose(datafilename);
    fclose(fp1);
  }
}

view ()

{
  printf("Is the file format rules or tests? ");
  scanf("%s", response);
  do
```

```
{
  printf("What is the name of your database file? ");
  scanf("%s", datafilename);
  if (mxopen(DATABASE, datafilename, "r"))
    {
      mxclose(datafilename);
      ch=1;
    }
   else
  {
    printf("Doesn't exist. \n");
    ch=0;
  }
} while (ch == 0);

menu_ok=1
while (menu_ok==1)
{
  a=getchar();
  system("clear");
  printf("\n");
  printf("            ~~~~~~~~~~~~~~~~~ VIEW MENU ~~~~~~~~~~~~~~~~~~\n");
  printf("            ~~~~~~~~~~~~~~~~~ --------- ~~~~~~~~~~~~~~~~~~\n");
  printf("            ~                                             ~\n");
  printf("            ~ 1)   Create an index                        ~\n");
  printf("            ~                                             ~\n");
  printf("            ~ 2)   Sort a specific field                  ~\n");
  printf("            ~                                             ~\n");
  printf("            ~ 3)   View the database                      ~\n");
  printf("            ~                                             ~\n");
  printf("            ~ 4)   Search for a database entry            ~\n");
  printf("            ~                                             ~\n");
  printf("            ~ 5)   Add a record to the database           ~\n");
  printf("            ~                                             ~\n");
  printf("            ~ 6)   Return to main menu                    ~\n");
  printf("            ~                                             ~\n");
  printf("            ~~~~~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
  printf("\n");
  printf("Input in the number, please ");
  answer=getchar();
  switch(answer)
  {
    case '1':   if (aindex<1)
                  fast_index(datafilename, response);
                 else
                {
                  printf("You already have %d index.  Do you want", aindex);
                  printf(" to D)elete and proceed or P)roceed? ");
                  scanf("%s", bindex);
                  if (toupper(bindex[0])=='P')
                    fast_index(datafilename, response);
                   else
                  {
                    printf("You have these indexes to delete: ");
                    for (i=1; i<=aindex; ++i)
                      printf("%d)  %s\n", i, indexed[i].name);
                    printf("\nType in the number ");
                    scanf("%d", &i);
```

```
                        strcpy(STRING, "drop index on ");
                        strcat(STRING, indexed[i].name);
                        mscall(DATABASE, STRING);
                        if (i<aindex)
                          for (ii=i; ii<=aindex-1; ii++)
                            strcpy(indexed[ii].name, indexed[ii+1].name);
                        -- aindex;
                        fast_index(datafilename, response);
                      }
                    }
                    break;
        case '2':   sort_rec(datafilename, response);
                    break;
        case '3':   show_record(datafilename, response);
                    break;
        case '4':   look_for(datafilename, response);
                    break;
        case '5':   add_record(datafilename, response);
                    break;
        case '6':   menu_ok=0;
                    break;
        default:    printf("Incorrect response.\n");
                    break;
      }
    }
    menu_ok=1;    /*prevents side effects in the main program*/
}

fast_index(filename, type)
char filename[30], type[30];

{
  char query[50];

  printf("What field do you want an index on:\n");
  if (type[0]=='r' || type[0]=='R')
    printf("Module, Cause, Effect, Type, or Precondition? ");
  if (type[0]=='t' || type[0]=='T')
    {
      printf("Name, Test, Parameter, Units, Qualifying Value, Min, \n");
      printf("Max, Cost, Instructions? ");
    }
  scanf("%s", query);
  strcpy(STRING, "create index on ");
  for (i=0; i<=strlen(query); ++i)
    if (isupper(query[i]))
      query[i]=tolower(query[i]);
  if ((strncmp(query, "prec", 4))==0)
    strcpy(query, "precond");
  if ((strncmp(query, "test", 4))==0)
    strcpy(query, "test_point");
  if ((strncmp(query, "quali", 5))==0)
    strcpy(query, "quall");
  strcat(STRING, filename);
  strcat(STRING, ".");
  strcat(STRING, query);
  mscall(DATABASE, STRING);
  ++aindex;
```

```
  strcpy(indexed[aindex].name, filename);
  strcat(indexed[aindex].name, ".");
  strcat(indexed[aindex].name, query);
}

sort_rec(filename, type)

char filename[30], type[30];

{
  char query[30], sorter[30], output2[10];
  int output;

  printf("What field are you sorting on:\n");
  if (type[0]=='r' || type[0]=='R')
    printf("Module, Cause, Effect, Type, or Precondition? ");
  if (type[0]=='t' || type[0]=='T')
    {
      printf("Name, Test, Parameter, Units, Qualifying Value, Min, \n");
      printf("Max, Cost, Instructions? ");
    }
  scanf("%s", query);
  printf("Do you want to sort ascending or descending? ");
  scanf("%s", sorter);
  printf("Do you want to output to the 1) screen or a 2) file? ");
  scanf("%d", &output);
  if (output==2)
  {
    printf("What is the file name? ");
    scanf("%s", output2);
  }
  strcpy(STRING, "select ");
  for (i=0; i<=strlen(query); ++i)
    if (isupper(query[i]))
      query[i]=tolower(query[i]);
  if ((strncmp(query, "prec", 4))==0)
    strcpy(query, "precond");
  if ((strncmp(query, "test", 4))==0)
    strcpy(query, "test_point");
  if ((strncmp(query, "quali", 5))==0)
    strcpy(query, "qual1");
  if (type[0]=='t' || type[0]=='T')
    strcat(STRING, "name, test_point, parameter, qual1");
  if (type[0]=='r' || type[0]=='R')
    strcat(STRING, "module, cause, effect");
  strcat(STRING, " from ");
  strcat(STRING, filename);
  strcat(STRING, " sort by ");
  strcat(STRING, query);
  strcat(STRING, " ");
  strcat(STRING, sorter);
  if (output == 2)
  {
    strcat(STRING, " into ");
    strcat(STRING, output2);
  }
  mscall(DATABASE, STRING);
}
```

```
look_for(filename, type)

char filename[30], type[30];

{
  char query[30], term[30];

  printf("\nWhat field are you searching on:\n");
  if (type[0]=='r' || type[0]=='R')
    printf("Module, Cause, Effect, Type, or Precondition? ");
  if (type[0]=='t' || type[0]=='T')
      printf("Name, Test, Parameter, Units? ");
  scanf("%s", query);
  for (i=0; i<=strlen(query); ++i)
    if (isupper(query[i]))
      query[i]=tolower(query[i]);
  if ((strncmp(query, "prec", 4))==0)
    strcpy(query, "precond");
  if ((strncmp(query, "test", 4))==0)
    strcpy(query, "test_point");
  printf("What term are you searching for? (Please place in quotes) ");
  scanf("%s", term);
  strcpy(STRING, "select from ");
  strcat(STRING, filename);
  strcat(STRING, " where ");
  strcat(STRING, query);
  strcat(STRING, " = ");
  strcat(STRING, term);
  mscall(DATABASE, STRING);
  printf("\nPress enter when ready.");
  answer=getchar();
}

show_record(filename, type)

char filename[30], type[30];

{
  char modisponse[10], new[75], xname[30];
  int field, count, d;

  previous=0;
  mxopen(DATABASE, filename, "u");
  if (type[0]=='r' || type[0]=='R')
  for(;;)
  {
    printf("      ***** RULES DATABASE *****\n\n");
    mxgetbegin (filename, CHARNIL);
    while (mxget ())
    {
      if (previous!=0)
      {
        previous=0;
        mxprev();
        mxprev();
      }
      strcpy(rulevar.module, mxgetvs("module"));
```

```
      strcpy(rulevar.frate, mxgetvs("frate"));
      strcpy(rulevar.cause, mxgetvs("cause"));
      strcpy(rulevar.effect, mxgetvs("effect"));
      strcpy(rulevar.type, mxgetvs("type"));
      strcpy(rulevar.precond, mxgetvs("precond"));
      printf("Module name: %s\n\nFrate: %s\nCause: %s\nEffect: %s      ",
             rulevar.module, rulevar.frate, rulevar.cause, rulevar.effect);
         printf("Type:  %s         Precondition:  %s\n",  rulevar.type,
rulevar.precond);
      printf("\nP)revious, N)ext, D)elete, M)odify, S)kip, or V)iew menu? ");
      scanf("%s", modisponse);
      if (toupper(modisponse[0])=='D')
      {
        printf("Are you sure? (Y/N) ");
        scanf("%s", modisponse);
        if (toupper(modisponse[0])=='Y')
          mxdel(filename);
         else
          mxprev();
      }
      if (toupper(modisponse[0])=='V')
      {
        mxgetend();
        mxclose(filename);
        break;
      }
      if (toupper(modisponse[0])=='S')
      {
        printf("If you want to skip a certain number of records ahead,\n");
        printf("press 1, if you know the module name, press 2 ");
        scanf("%d", &d);
        if (d==1)
        {
          printf("How many would you like to skip? ");
          scanf("%d", &d);
          if (d>0)
            for (count=0; (count<d && mxget()); ++count);
           else
            printf("Positive numbers only, please.\n");
          mxprev();
        }
        if (d==2)
        {
          printf("What is the module name? ");
          scanf("%s", xname);
          while((strcmp(xname, rulevar.module)) && mxget())
            strcpy(rulevar.module, mxgetvs("module"));
          mxprev();
        }
      }
      if (toupper(modisponse[0])=='P')
        previous=1;
      if (toupper(modisponse[0])=='M')
      {
        printf("1) Module\n2) Cause\n3) Effect\n4) Type\n5) Precondition\n6)
Frate");
        printf("\nEnter the number of the field you want to modify: ");
        scanf("%d", &field);
```

```
        printf("What do you want put in its place? ");
        scanf("%s", new);
        if (field==1)
          mxputvs("module", new);
        if (field==2)
          mxputvs("cause", new);
        if (field==3)
          mxputvs("effect", new);
        if (field==4)
          mxputvs("type", new);
        if (field==5)
          mxputvs("precond", new);
        if (field==6)
          mxputvs("frate", new);
        mxput(filename);
        mxprev();
      }
    }
    break;
  }

  if (type[0]=='t' || type[0]=='T')
  for(;;)
  {
    printf("     ***** TEST DATABASE *****\n\n");
    mxgetbegin(filename, CHARNIL);
    while (mxget ())
    {
      if (previous!=0)
      {
        previous=0;
        mxprev();
        mxprev();
      }
      strcpy(testvar.name, mxgetvs("name"));
      strcpy(testvar.test_point, mxgetvs("test_point"));
      strcpy(testvar.parameter, mxgetvs("parameter"));
      strcpy(testvar.units, mxgetvs("units"));
      strcpy(testvar.qual_val1, mxgetvs("qual1"));
      strcpy(testvar.min, mxgetvs("min"));
      strcpy(testvar.max, mxgetvs("max"));
      strcpy(testvar.cost, mxgetvs("cost"));
      strcpy(testvar.prereq, mxgetvs("prereq"));
      strcpy(testvar.instruction, mxgetvs("instruction"));
      strcpy(testvar.test, mxgetvs("test"));
      strcpy(testvar.type, mxgetvs("type"));
      printf("Name %s\nTest Point %s\nParameter %s\nUnits %s\n", testvar.name,
             testvar.test_point, testvar.parameter, testvar.units);
      printf("Qualifying Values %s Min %s    Max %s\nCost %s\n",
             testvar.qual_val1, testvar.min, testvar.max, testvar.cost);
      printf("Prerequisites %s\nInstruction %s\nType %s\nFocal Module %s\n",
                  testvar.prereq,  testvar.instruction,  testvar.test,
testvar.type);
      printf("\nP)revious, N)ext, D)elete, M)odify, S)kip, or V)iew menu? ");
      scanf("%s", modisponse);
      if (toupper(modisponse[0])=='D')
      {
        printf("Are you sure? (Y/N) ");
```

```
  scanf("%s", modisponse);
  if (toupper(modisponse[0])=='Y')
    mxdel(filename);
   else
    mxprev();
}
if (toupper(modisponse[0])=='V')
{
  mxgetend();
  mxclose(filename);
  break;
}
if (toupper(modisponse[0])=='S')
{
  printf("If you want to skip a certain number of records ahead,\n");
  printf("press 1, if you know the test name, press 2 ");
  scanf("%d", &d);
  if (d==1)
  {
    printf("How many would you like to skip? ");
    scanf("%d", &d);
    if (d>0)
      for (count=0; (count<d && mxget()); ++count);
     else
      printf("Positive numbers only, please.\n");
    mxprev();
  }
  if (d==2)
  {
    printf("What is the test name? ");
    scanf("%s", xname);
    while((strcmp(xname, testvar.name)) && mxget())
      strcpy(testvar.name, mxgetvs("name"));
    mxprev();
  }
}
if (toupper(modisponse[0])=='P')
  previous=1;
if (toupper(modisponse[0])=='M')
{
  printf("1) Name\n2) Test Point\n3) Parameter\n4) Units\n");
  printf("5) Qualifying Values\n6) Min\n7) Max\n8) Cost\n");
  printf("9) Prereq\n10) Instruction\n11) Type\n12) Focal Module\n");
  printf("Enter the number of the field you want to modify: ");
  scanf("%d", &field);
  printf("What do you want to put in its place? ");
  scanf("%s", new);
     /* To be able to read in data without the program misinterpreting
        spaces as enters, perhaps read in character by character and
        terminate with enter.  Possible functions, isalnum(), isspace(),
        ispunct(), isprint()*/
  if (field==1)
    mxputvs("name", new);
  if (field==2)
    mxputvs("test_point", new);
  if (field==3)
    mxputvs("parameter", new);
  if (field==4)
```

```
            mxputvs("units", new);
          if (field==5)
            mxputvs("qual1", new);
          if (field==6)
            mxputvs("min", new);
          if (field==7)
            mxputvs("max", new);
          if (field==8)
            mxputvs("cost", new);
          if (field==9)
            mxputvs("prereq", new);
          if (field==10)
            mxputvs("instruction", new);
          if (field==11)
            mxputvs("test", new);
          if (field==12)
            mxputvs("type", new);
          mxput(filename);
          mxprev();
        }
      }
      break;
    }
}

create ()

{
  printf("Is the file format rules or tests? ");
  scanf("%s", response);
  do
  {
    printf("What do you want to name the new file? ");
    scanf("%s", datafilename);
    if (mxopen(DATABASE, datafilename, "u"))
      {
        ch=1;
        printf("That file already exists.  Pick another.\n");
        mxclose(datafilename);
      }
       else
                  /*here's where we create the format*/
      if (response[0]=='r' || response[0]=='R')
      {
        strcpy(STRING, "create ");
        strcat(STRING, datafilename);
        strcat(STRING, " from rules_stand;");
                  /*this calls the file rules_stand from EMPRESS that
                     copies the data table information to the new file*/
        mscall(DATABASE, STRING);
        ch=0;
      }
       else
      {
        strcpy(STRING, "create ");
        strcat(STRING, datafilename);
        strcat(STRING, " from test_stand;");
        mscall(DATABASE, STRING);
```

```
        ch=0;
      }
  } while (ch != 0);

  add_record(datafilename, response);
}

add_record(filename, type)

char filename[30], type[30];

{
  answer1[0]='n';
  mxopen(DATABASE, filename, "u");
  if (type[0]=='t' || type[0]=='T')  /*test database*/
    while (answer1[0]=='N' || answer1[0]=='n')
    {
      getchar();
      printf("What is the name? ");
      gets(testvar.name);
      printf("What is the test point? ");
      gets(testvar.test_point);
      printf("What is the parameter? ");
      gets(testvar.parameter);
      printf("What are the units?\n ");
      gets(testvar.units);
      printf("What are the qualifying values? (list ok bad, etc) ");
      gets(testvar.qual_val1);
      printf("What is the min (Press <Enter> if none)? ");
      gets(testvar.min);
      printf("What is the max (Press <Enter> if none)? ");
      gets(testvar.max);
      printf("What is the cost? ");
      gets(testvar.cost);
      printf("What are the prerequisites (Put a space between each)? \n");
      gets(testvar.prereq);
      printf("What are the instructions? (Press <Enter> if none)?\n");
      gets(testvar.instruction);
      printf("What is the type (Press <Enter> if none)?\n");
      gets(testvar.test);
      printf("What is the Focal Module? ");
      gets(testvar.type);
      printf("\n");
      printf("Name %s\nTest Point %s\nParameter %s\nUnits %s\n", testvar.name,
             testvar.test_point, testvar.parameter, testvar.units);
      printf("Qualifying Values %s Min %s    Max %s\nCost %s\n",
             testvar.qual_val1, testvar.min, testvar.max, testvar.cost);
      printf("Prerequisites %s\nInstruction %s\nType %s\nFocal Module %s\n",
                    testvar.prereq, testvar.instruction, testvar.test,
testvar.type);
      printf("\n");
      printf("Is this correct (Y/N)? ");
      scanf("%s", answer1);
      if (answer1[0]=='y' || answer1[0]=='Y')
      {
        mxputvs("name", testvar.name);
        mxputvs("test_point", testvar.test_point);
        mxputvs("parameter", testvar.parameter);
```

```
            mxputvs("units", testvar.units);
            mxputvs("qual1", testvar.qual_val1);
            if (testvar.min[0] != '0')
              mxputvs("min", testvar.min);
            if (testvar.max[0] != '0')
              mxputvs("max", testvar.max);
            mxputvs("cost", testvar.cost);
            mxputvs("prereq", testvar.prereq);
            if (testvar.instruction[0] != '0')
              mxputvs("instruction", testvar.instruction);
            if (testvar.test[0] != '0')
              mxputvs("test", testvar.test);
            mxputvs("type", testvar.type);
            mxadd(filename);
            mxclose(filename);
          }
        }
      answer1[0]='N';
      if (type[0]=='r' || type[0]=='R')  /*rule database*/
        while (answer1[0]=='N' || answer1[0]=='n')
        {
          getchar();
          printf("What is the module name? ");
          gets(rulevar.module);
          printf("What is the cause? ");
          gets(rulevar.cause);
          printf("What is the effect? ");
          gets(rulevar.effect);
          printf("What is the type? ");
          gets(rulevar.type);
          printf("What is the precondition? ");
          gets(rulevar.precond);
          printf("\n");
          printf("Module name: %s\n\nCause: %s\nEffect: %s      ", rulevar.module,
                  rulevar.cause, rulevar.effect);
              printf("Type:  %s        Precondition:  %s\n",  rulevar.type,
rulevar.precond);
          printf("\n");
          printf("Is this correct (Y/N)? ");
          scanf("%s", answer1);
          if (answer1[0]=='y' || answer1[0]=='Y')
          {
            mxputvs("module", rulevar.module);
            mxputvs("cause", rulevar.cause);
            mxputvs("effect", rulevar.effect);
            mxputvs("type", rulevar.type);
            mxputvs("precond", rulevar.precond);
            mxadd(filename);
            mxclose(filename);
          }
        }
    }
```